RL-TR-96-177
Final Technical Report
December 1996

SIGNAL PROCESSING FOR RADAR TARGET TRACKING AND IDENTIFICATION

Washington University

Joseph A. O'Sullivan, Donald L. Snyder,
and Michael I. Miller

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

19970305 040

DTIC QUALITY INSPECTED 3

Rome Laboratory
Air Force Materiel Command
Rome, New York

PII Redacted

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-96-177 has been reviewed and is approved for publication.

APPROVED: Vincent Vannicola

VINCENT VANNICOLA
Project Engineer

FOR THE COMMANDER: Gary D. Barmore

GARY D. BARMORE, Major, USAF
Deputy Director
Surveillance & Photonics Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL/OCSS, 26 Electronic Pky, Rome, NY 13441-4514. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)

2. REPORT DATE
December 1996

3. REPORT TYPE AND DATES COVERED
Final Dec 91 - Jun 95

4. TITLE AND SUBTITLE
SIGNAL PROCESSING FOR RADAR TARGET TRACKING AND IDENTIFICATION

5. FUNDING NUMBERS
C - F30602-92-C-0004
PE - 62702F
PR - 4594
TA - 00
WU - 5N

6. AUTHOR(S)
Joseph A. O'Sullivan, Donald L. Snyder, and Michael I. Miller

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Washington University
Electronic Systems and Signals Research Laboratory
Department of Electrical Engineering
St. Louis, MO 63130

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory/OCSS
26 Electronic Pky
Rome, NY 13441-4514

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-96-177

11. SUPPLEMENTARY NOTES
Project Engineer: Vincent Vannicola/OCSS/(315) 330-2861

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The work reported here has yielded significant new results in airborne target identification. The framework of the approach consists of a joint target tracking and recognition scenario, and a general mathematical approach for addressing this and similar problems. By tracking the kinematics of target, it is shown how the software is better able to focus on which class of orientations and aspect angles to start the template matching for the identification phase. An S-band radar operating in a wideband mode along with an optical sensor constituted the two high resolution data resources. Simulations were also developed, including outputs from X-patch, for each of these sensors. Demonstrations of the software running are also presented.

14. SUBJECT TERMS Airborne targets, Joint target tracking, recognition scenario, high resolution sensors, S-Band radar, wideband mode, optical sensor

15. NUMBER OF PAGES
492

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT
UNCLASSIFIED

18. SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

19. SECURITY CLASSIFICATION OF ABSTRACT
UNCLASSIFIED

20. LIMITATION OF ABSTRACT
UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

Table of Contents

Abstract 2

I. Introduction 2

II. Activity 3

A. Personnel 3

B. Collaboration 3

C. Degrees 4

D. Visits 5

III. Papers and Presentations 6

IV. ATR Using HRR Data 9

V. Software 55

Appendix A: A. Srivastava papers and thesis A-1

Appendix B: Various papers B-1

Appendix C: K. Cecilia Du thesis C-1

DTIC QUALITY INSPECTED 3

Abstract

In December 1991, Rome Laboratory awarded the contract "Signal Processing for Radar Target Tracking and Identification," contract number F30602-92-C-0004. The work we have performed on this contract has yielded significant new results in automatic target recognition. The framework of the approach developed consists of a joint target tracking and recognition scenario, and a general mathematical approach for addressing this and similar problems. The work performed for Rome Laboratory has led to applications in other areas including related military problems and medical problems. The dual-use nature of many of the algorithms and the mathematical basis are of fundamental importance and must be pursued vigorously in future research.

The two high resolution sensors upon which we focused were an S-band radar operating in a wideband mode and an optical sensor. Simulations were developed for each of these sensors and demonstrations of the software running were given at the time of the oral final report on June 12, 1995.

I. Introduction

The general area studied has been multitarget tracking and recognition using data from multiple sensors. Central to the view taken in our work is the observation that fusion of data from multiple sensors requires tracking of the target positions and orientations relative to all of the sensors involved. That is, in order to combine data from multiple sensors, an accurate model of the joint data sets is required; this model must include the relative motions of all sensors and targets. Note that any improvements in the orientation estimates for the targets can lead to improved recognition performance, since recognition itself often involves orientation estimation. Also, improved recognition can lead to better tracking performance by including more accurate target motion models and improved orientation estimates.

A detailed description of the problem studied under contract number F30602-92-C-0004 first appeared in our progress report dated July 29, 1992. The problem consists of the simultaneous tracking and recognition of targets using multisensor data. Our focus has been on radar sensors and optical imaging sensors.

The problem may be described as follows. Assume that there are two sensors available collecting measured data from an aircraft. One sensor is a tracking radar that provides estimates of the target position (usually in range, azimuth, and elevation coordinates). A second sensor provides high range resolution measurements of the target. From these two data sets, the algorithm being developed will attempt to track and recognize the target. We assume that the target is one of a finite number of targets, and that we have a detailed description of each of the targets. The description consists of a dynamical model for the motion of the target and a surface facet model from which high range resolution profiles may be computed. The model is used to generate predicted measurements.

By developing a single joint model for the tracking and range profile data, we are able to improve the performance of both tracking and recognition. The tracking is improved because the high resolution sensor provides orientation information. The orientation information may be used in the tracker to generate better predictions for the next target position. The recognition is improved by the joint model because the dynamical model used in the tracker includes orientation dynamics that may be used, along with previous orientation estimates, to generate better predictions for the next target orientation than if the dynamics were not included.

One tool used is the theory of deformable templates. The templates for this problem are the models for the targets consisting of both the dynamics and the surface facet model. Perhaps the most important paper published to date in the area of deformable templates is by Miller and Grenander; the work leading to this paper was supported in part by this contract. A second tool is the theory of jump-diffusions. Our algorithms are based on jump-diffusions. A theoretical description of the use of jump-diffusions in algorithm development is in the Master's thesis of Anuj Srivastava which is included in Appendix A. This thesis constitutes a large part of this report.

II. Activity

Detailed summaries of the activity on this contract have been given in previous progress reports and in annual reports. This section gives some of the highlights of the activity, concentrating on activity since September 1994.

A. Personnel

The personnel at Washington University directly supported by this contract include faculty and students. They have met regularly for the duration of this contract. Those supported by the contract included:

- Faculty:
  - Prof. Donald L. Snyder
  - Prof. Joseph A. O'Sullivan
  - Prof. Michael I. Miller
- Students:
  - Anuj Srivastava
  - K. Cecilia Du
  - Brian Barber
  - Steve Jacobs
  - Robert Teichman
  - Nicholas Cutaia
  - Chandrakanth Gowda
  - Todd Ellebracht
  - Mohammad Faisal
  - Jason August
  - Mark Foltz

B. Collaborations

One aspect of the work which we feel was instrumental for the successful completion of the work on this contract is the cross-fertilization of ideas with related research projects in the department of Electrical Engineering at Washington University. We invited to our research meetings other individuals whose research interests are closely related to the ideas in this project. These individuals attended meetings regularly and included:

Faculty:
- Prof. Daniel R. Fuhrmann

Students:
- Jonathan Locker, Senior undergraduate
- Debra Michal, fourth year graduate
- Asif Chinwalla, second year graduate
- Aaron Lanterman, first year graduate

Some of these individuals not directly supported contributed materially to the results reported here and in other progress reports. For example, some of them have co-authored papers resulting from this research.

Another significant collaboration has been with Prof. Ulf Grenander at Brown University. Prof. Grenander is one of the most outstanding applied mathematicians of this century. Prof. M. I. Miller visited Brown University regularly over the duration of this project. Prof. Grenander visited Washington University during April 1995, a visit supported in part by this contract. Profs. Miller and Grenander wrote a significant joint paper, "Representation of Knowledge in Complex Systems," Journal of the Royal Statistical Society, Series B, Volume 56, Number 4, 1994, pp. 549-603. This paper was read before the Society and the paper includes discussion. It summarizes much of the mathematical foundations of our approach and copies have been provided to Rome Laboratory in the past.

C. Degrees

One of the primary benefits of supporting research at a university is in the human resource area. The development of graduates with skills necessary to contribute in areas critical to national defense is extremely important. The students working on this project are all skilled in automatic target recognition and have the kind of training so needed in the defense industry and associated government laboratories. One measure of this devlopment is by the degrees students working on this project have received over the course of this contract. These degrees are listed below:

Brian Barber	MSEE
Robert Teichman	MSEE
Cecilia Du	MSEE
Anuj Srivastava	MSEE
Todd Ellebracht	BSEE
Jon Locker	BSEE
Aaron Lanterman	MSEE
Asif Chinwalla	MSEE
Debra Michal	D.Sc.

In addition, several students are near completion of degrees:

Anuj Srivastava	D.Sc., Julyy 1996
Steve Jacobs	D.Sc., Aug. 1996
Nicholas Cutaia	D.Sc., Dec.. 1995
Mohammad Faisal	MSEE, May 1995
Mark Foltz	BSEE, May 1996

D. Visits

On January 17, 1992, Profs. Snyder, O'Sullivan, and Miller visited Rome Laboratory. The meeting was a kick-off meeting for the project. They summarized their past research efforts as they related to the anticipated research on this project. This meeting provided a valuable opportunity to meet individuals at Rome Laboratory and to tour the facilities.

On April 10, 1992, Drs. V. Vannicola and D. Benincasa from Rome Laboratory visited Washington University. They participated in a research meeting and toured our facilities. At the research meeting, all students and faculty briefed the visitors on progress. After the meeting, Dr. V. Vannicola provided additional information on Rome Laboratory facilities.

On December 4, 1992 Profs. O'Sullivan, Snyder, and Miller visited Rome Laboratory to report the first year's activity. This report was given both at Rome Laboratory and in a luncheon talk before the Mohawk Valley Chapter of the Signal Processing Society.

Two students working on this project, Anuj Srivastava and Robert Teichman, visited Rome Laboratory in May 1993. This visit was in conjunction with the Dual-Use conference in Utica. In addition to their presentation at the conference, these students visited Rome Laboratory and delivered a videotape showing jump-diffusion algorithms running on angle data. That is, instead of running on a typical Cartesian coordinate system $\mathbf{R}^n$, the jump-diffusion algorithm performed random searches over angle data (that have a compact domain with periodic boundaries).

Brian Barber from Washington University spent the summer of 1993 at Rome Laboratory. This visit resulted in Rome Laboratory providing some real data collected from their S-Band radar operating in the wideband mode. Brian provided an important bridge between our research efforts and the efforts of individuals at Rome Laboratory. In particular, Brian became extremely knowledgeable about the available data, he developed software to interface between the real data and our existing simulations, and he learned XPATCH. His summer work was very successful.

Rome Laboratory visited Washington University on May 26 and 27, 1994. The visiting personnel included B. Rydelek, V. Vannicola, and J. Capraro. Various discussions of the project took place including demonstrations of the software developed to date.

Profs. Miller and O'Sullivan visited Rome Laboratory on November 14, 1994. They gave a report on the third year's activity and showed a video of the current software running. They discussed the plans for the remaining few months of the project and for potential follow-on efforts.

Profs. Snyder and O'Sullivan visited Rome Laboratory on June 12, 1995 to give the final oral report for this project. The report consisted of a one hour presentation using the Powerpoint presentation tool along with a remote demonstration of the computations developed on this project. This remote demonstration was the first of its kind by the research group. The complete software package (see Section VI for a discussion of the software and a complete program listing) was shown running. One aspect of the demonstration was a remote visualization of a distributed computation; the computations were performed on various computers at Washington University in St. Louis, including the Maspar 12000 (the 128 by 128 massively parallel array of processors). The results of the computations were transmitted across the internet and visualized on a Silicon Graphics computer in the computer facility at Rome Laboratory. This demonstration showed a recognition algorithm running on high resolution radar data. Thus, the major goal of the research on this project was achieved and demonstrated to Rome Laboratory personnel.

III. Papers and Presentations

The primary methods of disseminating the results of research are through reports such as this, through presentations at conferences and workshops, and through the publication of papers describing the results in detail. The faculty and students presented the results of the research at many important conferences and workshops over the duration of this project. Also, several journal papers were submitted. Most of these are summarized below.

- The papers and presentations:

1. M. I. Miller and U. Grenander, "Representation of Knowledge in Complex Systems," Journal of the Royal Statistical Society, Series B, Volue 56, Number 4, 1994, pp. 549-603. Copies of this paper have been provided to Rome Laboratory. This paper is one of the most important theoretical contributions to this area in the past few years. It describes the method of deformable templates and the use of jump-diffusion algorithms to do inference.
2. J. A. O'Sullivan, M. I. Miller, A. Srivastava, and D. L. Snyder, "Tracking Using a Random Sampling Algorithm," 12th World Congress of the International Federation of Automatic Control, Sydney, Australia, July 1993. This paper is part of Appendix B of this report.
3. J. A. O'Sullivan, K. C. Du, R. S. Teichman, M. I. Miller, D. L. Snyder, and V. C. Vannicola, "Radar Target Recognition Using Shape Models," *Proceedings of the 30th Annual Allerton Conference on Communication, Control, and Computing,* Urbana, IL, October 1992. This paper, which was part of the first year's efforts is included here for completeness as part of Appendix B.
4. A. Srivastava, N. Cutaia, M. I. Miller, J. A. O'Sullivan, and D. L. Snyder, "Multi-Target Narrowband Direction Finding and Tracking Based on Motion Dynamics," *Proceedings of the 30th Annual Allerton Conference on Communication, Control, and Computing,* Urbana, IL, October 1992. This paper, which was part of the first year's efforts is included here for completeness as part of Appendix B.
5. J. A. O'Sullivan, "A New Wideband Radar Imaging Algorithm," submitted to the *IEEE Trans. on Image Processing,* 1993. Copies of this paper have been provided to Rome Laboratory in the past.
6. J. A. O'Sullivan, K. C. Du, R. S. Teichman, M. I. Miller, D. L. Snyder, and V. C. Vannicola, "Reflectivity Models for Radar Target Recognition," SPIE International Symposium on OE/Aerospace Sensing, Orlando, April 1993. This paper is part of

Appendix B of this report.

7. M. I. Miller, R. Teichman, A. Srivastava, J. A. O'Sullivan, and D. L. Snyder, "Jump-Diffusion Processes for Automated Tracking-Target Recognition," *Proceedings of the 27th Annual Conference on Information Sciences and Systems,* Johns Hopkins University, Baltimore, March 1993, pp. 617-622. This paper is part of Appendix B of this report.

8. J. A. O'Sullivan, "Likelihood Models for Sensor Fusion," Invited Talk, Army Research Office Information Fusion Workshop, Harper's Ferry, WV, June 1993.

9. K. C. Du, "Range Profile Prediction in Radar Target Recognition," M.S. Thesis, Washington University, September 1993. This thesis is Appendix C of this report. This research focused on the development of computationally efficient radar signature prediction tools for incorporation in radar target identification algorithms.

10. R. Teichman, A. Srivastava, M. I. Miller, J. A. O'Sullivan, and D. L. Snyder, "Automated Tracking and Target Recognition Algorithms Using Jump-Diffusion Processes," Presented at the 1993 IEEE Mohawk Valley Section Dual-Use Technologies and Applications Conference, Utica, NY, May 1993.

11. A. Srivastava, R. S. Teichman, and M. I. Miller, "Target Tracking and Recognition Using Jump-Diffusion Processes," ARO's 11th Army Conf. on Applied Mathematics and Computing, June 8-11, 1993, Carnegie Mellon university, Pittsburgh. This reprint is included as part of Appendix B of this report.

12. A. Srivastava, R. S. Teichman, M. I. Miller, D. L. Snyder, and J. A. O'Sullivan, "Jump-Diffusion Based Sampling Algorithm for Target Tracking and Recognition," 27th Asilomar Conf. on Signals, Systems, and Computers, November 1-3, 1993, Pacific Grove, CA. This reprint is included as part of Appendix B of this report.

13. J. A. O'Sullivan, S. P. Jacobs, M. I. Miller, and D. L. Snyder, "A Likelihood-Based Approach to Joint Target Tracking and Identification," 27th Asilomar Conf. on Signals, Systems, and Computers, November 1-3, 1993, Pacific Grove, CA. This reprint is included as part of Appendix B of this report.

14. A. Srivastava, "Automated Target Tracking and Recognition Using Jump-Diffusion Processes," M.S. Thesis, Washington University Department of Electrical Engineering, December 1993. This thesis is included as part of Appendix A of this report.

15. A. Srivastava, M. I. Miller, U. Grenander, "Multiple Target Direction of Arrival Tracking," *IEEE Transactions on Signal Processing,* Vol. 43, No. 5, May 1995, pp. 1282-1285. Copies of this paper have been provided to Rome Laboratory in the past.

16. M. I. Miller, A. Srivastava, and U. Grenander, "Conditional-Mean Estimation via Jump-Diffusion Processes in Multiple Target Tracking/Recognition," to appear in *IEEE Transactions on Signal Processing,* 1995. This paper is included as part of Appendix A of this report.

17. J. A. O'Sullivan, "A Unitary Algorithm for Wideband Radar Imaging," 1992 Conference on Information Sciences and Systems, Princeton University, March 18-20, 1992. Copies of this paper have been provided to Rome Laboratory in the past.

18. J. Locker, "Estimated Range Profiles for Radar Recognition Project," Final Report for Undergraduate Reaserch Project, Department of Electrical Engineering,

Washington Unviersity, St. Louis, 1992.

19. J. A. O'Sullivan, P. Moulin, D. L. Snyder, and D. G. Porter, "An Application of Estimation Theoretic Radar Imaging," *International Journal on Imaging Systems and Technology,* 1993. Copies of this paper have been provided to Rome Laboratory in the past.
20. A. Srivastava, M. I. Miller, U. Grenander, "Lie Group Parameterization for Dynamics Based Prior in ATR," reprint. This paper is presented as part of Appendix B to this report.
21. N. J. Cutaia and J. A. O'Sullivan, "Automatic Target Recognition Using Kinematic Priors," Proceedings CDC, December 1994, pp. 3303-3307. This paper describes methods for target classification based on dynamics alone. That is, the kinematics are used as the only discriminator. This paper is included as part of Appendix B to this report.
22. J. A. O'Sullivan, "Roughness Penalties for Image Estimation," Invited Talk, Research Group Seminar at MIT, March 10, 1995.

IV. ATR Using HRR Data

Data from a High Range Resolution (HRR) radar varies with target type and orientation. This document investigates this variation and the implications for performing orientation estimation and target identification. A new model for radar reflectivity is developed. The data variation due to target orientation is quantitatively evaluated by simulation. A simple test of the target-specific nature of actual data is performed. Future work is proposed.

1 Introduction

The problem of recognition of remote objects from sensor data has a long history, and has received particular attention in recent years. A variant of particular interest is the identification of aircraft targets from their radar reflections. While a great many articles have been published describing algorithms for identification, these algorithms generally require that the orientation of the aircraft during illumination by the radar be known or estimated. Successful orientation estimation is dependent on the behavior of the reflected radar signal as the target orientation changes. The purpose of this document is to investigate the nature of radar data for different targets and varying orientation and to discuss the implications for successful implementation of identification and orientation estimation tasks.

Consider the following environment for radar-based recognition of aircraft targets using an automated system. Some number M of aircraft are flying in the vicinity of a multiplicity of radar sensors. It is desired to detect, track, and identify each of the aircraft based on the complete history of measurements from the sensors. Typically, the sensors will include a low-frequency radar system explicitly designed for detection and tracking; one example of this would be a single centrally located radar transmitter combined with a cross array of passive receivers. Such a system can be used to determine the number of targets in the scene and their positions at any time. Once a target has been located, a high range resolution radar is steered in the direction of this target, and reflections resulting from an appropriately chosen transmitted signal are recorded. Based on the complete history of measurements from all sensors, it is desired to estimate for each target sequences of those parameters pertinent to the behavior of a dynamic aircraft: positions, velocities, orientations, rotation rates and aircraft type.

While it is conceivable that one could identify an aircraft without estimating all the associated dynamic parameters (e.g. visual identification of commercial aircraft by a human observer does not require estimation of rotation rates) it is clear that such estimation is crucial to successful identification by an automated system. Position estimates are required for steering of the HRR radar system. As orientation changes typically precede changes in aircraft acceleration, orientation estimation can aid in predicting the future position and velocity of the target. If the aircraft type is known, a dynamical model for this aircraft can be used to predict its behavior in flight. Also, the observed flight behavior may be indicative of the aircraft type. As a result, it is believed that any successful implementation of a system for Automatic Target Recognition will involve joint estimation of all these parameters from the available data.

This being the case, it is nonetheless true that most proposed systems for ATR do not consider the problem in this holistic context. The following is a partial list of simplifications to the ATR problem which allow for analysis of the performance of individual estimation tasks.

- It is known that a single target is present. Given a complete history of measurements from all sensors, estimate the positions, velocities, orientations, rotation rates, and aircraft type for this target.
- Only HRR data is available, and this data is assumed independent of position, velocity, and rotation rate. Given a series of range profiles, estimate the orientation and aircraft type for a single target.
- Only one range profile is available. From this single measurement, estimate the orientation and aircraft type for a single target.
- The orientation of the target is known or has been estimated by other means. From a single range profile, identify the aircraft type.
- The aircraft type of the target is assumed known. From a single observation of the range profile, estimate the orientation of the target.
- The orientation of the target is known, and it is also known that the target is one of two possible aircraft types. From a single range profile, determine which of the two aircraft types produced the observation.

It is interesting to note that the vast majority of proposed algorithms for ATR have addressed only the last of these simplified problems. Additionally, the data used to test these algorithms are frequently collected from scaled models in a compact radar range. While these results therefore do not impinge directly on the expected performance of algorithms which seek to solve the complete ATR problem through joint estimation, they have helped to identify important issues relating to the successful completion of smaller tasks.

In order to more clearly illustrate the ATR problem and important elements of its solution, a mathematical development is presented. We begin by considering the first of the simplified problems listed above: it is known that a single target is present, and it is desired to track and identify this target. A series of measurements are received from each of the sensors; let $r_k(t)$ denote the kth vector of received waveforms. Let the complete state vector for the target at time of the kth measurement, including positions, velocities, orientations, rotation rates and target type, be denoted by ξ_k. The received signal is the sum of a known, deterministic signal portion that is expressly dependent on the state ξ_k and a noise term

$$r_k(t) = s(t;\xi_k) + w_k(t) \tag{1}$$

where $w_k(t)$ is a sample waveform from a complex white Gaussian random process with mean 0 and intensity N_0. Let R_k and Ξ_k denote, respectively, the collection of received signal waveforms and state vectors up to the time of the kth measurement

$$\begin{aligned} R_k &= \{r_1(t), r_2(t), \ldots, r_k(t)\} && (2)\\ \Xi_k &= \{\xi_1, \xi_2, \ldots, \xi_k\}\,. && (3) \end{aligned}$$

Having received the kth measurement it is desired to estimate the state history Ξ_k which resulted in the measurement history R_k. Given the above statistical model for our measurements, the estimate we seek is the value of Ξ_k which maximizes the posterior probability density conditioned on the measurements

$$\hat{\Xi}_{k,\mathrm{MAP}} = \mathrm{argmax}_{\Xi_k}\, p\left(\Xi_k | R_k\right). \tag{4}$$

The posterior density is decomposed through Bayes' rule

$$p\left(\Xi_k | R_k\right) = \frac{p\left(\Xi_k\right) L\left(R_k | \Xi_k\right)}{L\left(R_k\right)}. \tag{5}$$

In this expression, $p\left(\Xi_k\right)$ is the prior density on the state vector history, $L\left(R_k|\Xi_k\right)$ is the data likelihood conditioned on the state history, and the denominator $L\left(R_k\right)$ is the joint likelihood of the measurements which is independent of the states and is therefore a constant with respect to maximization over Ξ_k.

The analysis is simplified by making the assumption that the states form a Markov chain: the probability of the next state ξ_{k+1} assuming any particular value is dependent only on the value of the current state ξ_k and not on any previous states. Thus the prior density may be written

$$p(\Xi_k) = p(\xi_0) \prod_{\ell=1}^{k} p(\xi_\ell | \xi_{\ell-1}) . \tag{6}$$

Through this assumption and the assumption, implicit in our data model, that, conditioned on knowledge of ξ_k, $r_k(t)$ is independent of previous measurements and states, the data likelihood may be written

$$L(R_k | \Xi_k) = \prod_{\ell=1}^{k} L(r_\ell | \xi_\ell) . \tag{7}$$

Thus, the posterior probability we seek to maximize may be written

$$p(\Xi_k | R_k) = \frac{p(\xi_0)}{L(R_k)} \prod_{\ell=1}^{k} p(\xi_\ell | \xi_{\ell-1}) L(r_\ell | \xi_\ell) . \tag{8}$$

An algorithm has been developed, as described in [7, 8, 13] which provides state estimates by sampling from this posterior density using Jump-Diffusion processes. The prior density is formed from the Newtonian force equations governing rigid body dynamics, and the data likelihood results from a statistical characterization of each of the sensors.

As mentioned previously, this document is primarily concerned with the estimation of target type and orientation from range profile data. Therefore, we now consider the fifth of the six simplified ATR problems: the target type is assumed known and we wish to estimate the orientation which resulted in the observation of a given range profile. The model for the range profile is similar to that used previously,

$$r_k(t) = s(t;\theta_k) + w_k(t), \tag{9}$$

where $s(t;\theta_k)$ is a known deterministic reflectivity profile which depends only on the orientation angles, collectively denoted as θ_k, and $w_k(t)$ is a sample waveform from a complex white Gaussian random process with mean 0 and intensity N_0. Proceeding as before,

$$\begin{aligned}
R_k &= \{r_1(t), r_2(t), \ldots, r_k(t)\} && (10)\\
\Theta_k &= \{\theta_1, \theta_2, \ldots, \theta_k\} && (11)\\
\hat{\Theta}_{k,\mathrm{MAP}} &= \mathrm{argmax}_{\Theta_k}\, p\left(\Theta_k | R_k\right) && (12)\\
p\left(\Theta_k | R_k\right) &\propto p\left(\Theta_k\right) L\left(R_k | \Theta_k\right) && (13)\\
&= p\left(\theta_0\right) \prod_{\ell=1}^{k} p\left(\theta_\ell | \theta_{\ell-1}\right) L\left(r_\ell | \theta_\ell\right). && (14)
\end{aligned}$$

In practice, the prior density on θ_ℓ may be further conditioned on knowledge of other states (e.g., the history of target positions). In any case, we will assume that this density has been provided to us, so that the quantity of interest is the likelihood function $L\left(r_k | \theta_k\right)$.

Let the received signal $r_k(t)$ be written as the sum

$$r_k(t) = \lim_{M\to\infty} \sum_{m=1}^{M} r_{k,m}\phi_m(t), \tag{15}$$

where $\{\phi_m(t)\}$ is a set of basis functions which form a complete orthonormal set, the coefficients $r_{k,m}$ are given by

$$\begin{aligned} r_{k,m} &= \int r_k(t)\phi_m(t)dt && (16)\\ &= \int s(t;\theta_k)\phi_m(t)dt + \int w_k(t)\phi_m(t)dt && (17)\\ &= s_m(\theta_k) + w_m && (18) \end{aligned}$$

and convergence of the limit is in the mean square sense. The truncation of the series representation for $r_k(t)$ to M terms is denoted $r_k^M(t)$. The likelihood function for $r_k^M(t)$ is given by

$$L\left(r_k^M|\theta_k\right) = \prod_{m=1}^{M} \frac{1}{\sqrt{2\pi N_0}} \exp\left(-\frac{1}{N_0} Re\left\{[r_{k,m} - s_m(\theta_k)][r_{k,m} - s_m(\theta_k)]^*\right\}\right). \tag{19}$$

Finally, the likelihood function for the received waveform $r(t)$ is defined as

$$L\left(r_k|\theta_k\right) = \lim_{M\to\infty} \frac{L\left(r_k^M|\theta_k\right)}{L\left(r_k^M|s(t;\theta_k) = 0\right)} \tag{20}$$

which yields

$$L\left(r_k|\theta_k\right) = \exp\left(\frac{2}{N_0} Re\left\{\int [r_k(t) - \frac{1}{2}s(t;\theta_k)][s(t;\theta_k)]^* dt\right\}\right). \tag{21}$$

Note that the received waveform $r_k(t)$ enters this expression through an inner product with the signal $s(t;\theta_k)$. In later sections we will be concerned with the variation of this inner product for small changes in the orientation θ_k, and properties of the variation which may be indicative of the potential performance of algorithms for orientation estimation.

The remainder of this document is organized as follows. Section 2 presents a review of the work of Shapiro in developing reflectivity models for laser radar. Section 3 contains a listing of several questions relating to radar-based ATR which remain unanswered. Section 4 contains some initial results toward answering these questions, including a model for radar reflectivity based on the theory of random arrays, analysis of simulated range profiles in terms of the difficulties encountered in orientation estimation, and analysis of real HRR data observed for several commercial aircraft. Finally, section 5 lists what conclusions can be drawn from our initial results and proposes further items to be completed.

2 Laser Radar Model

Shapiro [10, 11, 12] considers illumination of a target by the focused beam of a coherent laser radar. The beam radius is assumed to be small on the scale of the target and an image is formed by making separate measurements of the target reflectivity for different transmitter aim points. High resolution range determination for each aim point allows for a partial 3-D characterization of the target.

The analysis begins with a completely deterministic representation of the reflectivity of a point on the target through the matrix

$$\overline{\overline{\mathbf{R}}}(\overline{\Omega}_r, \omega_r; \overline{\Omega}_i, \omega_i; \omega_o). \tag{22}$$

This matrix is functionally parameterized by the incident and reflected direction vectors $\overline{\Omega}_i$ and $\overline{\Omega}_r$, the incident and reflected modulation frequencies ω_i and ω_r and the optical carrier frequency ω_o. The elements of this matrix contain the scattering amplitudes for the four combinations of horizontal and vertical polarizations of the transmitted and received waveforms. The Fourier component of the field $\overline{\mathcal{E}}_r$ reflected in the $\overline{\Omega}_r$ direction is given by the linear superposition of reflections arising from different incident directions and frequencies,

$$\overline{\mathcal{E}}_r(\overline{\Omega}_r, \omega_r) = \int \frac{d\omega_i}{2\pi} \int d\overline{\Omega}_i \left(\frac{\omega_o + \omega_i}{2\pi c} \right) \overline{\overline{\mathbf{R}}}(\overline{\Omega}_r, \omega_r; \overline{\Omega}_i, \omega_i; \omega_o) \overline{\mathcal{E}}_i(\overline{\Omega}_i, \omega_i). \tag{23}$$

Application of this model to the case of a uniformly polarized monochromatic incident field illuminating a target that is unresolved and stationary leads to familiar expressions for received power and radar cross section.

The complicated nature of this deterministic model makes determination of the $\overline{\overline{\mathbf{R}}}$-matrix of a complex target impractical. A simplification is suggested by which a deterministic quantity like the $\overline{\overline{\mathbf{R}}}$-matrix is used to describe the reflectance properties associated with the gross shape of a target, and the effects of the micro-scale features on the target surface are given a statistical characterization. Under the conditions of a stationary unresolved target and a monostatic radar system, the complex envelopes of the incident and reflected fields $\overline{\mathbf{E}}_i$ and $\overline{\mathbf{E}}_r$ are related through a multiplicative target model

$$\overline{\mathbf{E}}_r(\overline{\rho}, t) = \mathbf{T}(\overline{\rho})\overline{\mathbf{E}}_\mathrm{i}(\overline{\rho}, \mathbf{t}), \tag{24}$$

where $\overline{\rho}$ is the transmitter aim point in the image plane and $\mathbf{T}(\overline{\rho})$ is the associated complex-field reflection coefficient. This quantity is decomposed into a deterministic component $\mathbf{T}_\mathbf{g}(\overline{\rho})$ which models the target as a specular reflector such as a polished surface and a random component $\mathbf{T}_\mathrm{s}(\overline{\rho})$ which characterizes the effect of the roughness of the target surface

$$\mathbf{T}(\overline{\rho}) = \mathbf{T}_\mathbf{g}(\overline{\rho})\mathrm{e}^{\mathrm{j}\theta} + \mathbf{T}_\mathrm{s}(\overline{\rho}). \tag{25}$$

Shapiro considers the glint component $\mathbf{T}_\mathbf{g}(\overline{\rho})$ to have a very narrow angular extent like that of a mirror, but any appropriate reflection pattern could be substituted for this term. The phase angle θ is modeled as a uniform random variable on $[0, 2\pi]$. The speckle component $\mathbf{T}_\mathrm{s}(\overline{\rho})$ is a complex-valued 0-mean Gaussian random process with autocorrelation function

$$E\left\{\mathbf{T}_\mathrm{s}(\overline{\rho}_1)\mathbf{T}_\mathrm{s}(\overline{\rho}_2)\right\} = 0, \tag{26}$$

$$E\left\{\mathbf{T}_\mathrm{s}(\overline{\rho}_1)\mathbf{T}^*_\mathrm{s}(\overline{\rho}_2)\right\} = \lambda^2 \mathcal{T}_f(\overline{\rho}_\infty)\delta(\overline{\rho}_\infty - \overline{\rho}_\in). \tag{27}$$

The bidirectional reflectance is defined as the ratio of reflected to incident radiance as a function of wavelength and the directions of incidence and reflection, and is related to $\mathbf{T}(\overline{\rho})$ through

$$\rho'(\lambda; \overline{f}_i; \overline{f}_r) = \left(\lambda^2 A_T\right)^{-1} E\left\{\left|\int d\overline{\rho} \exp\left[j2\pi(\overline{f}_i - \overline{f}_r)\cdot\overline{\rho}\right]\mathbf{T}(\overline{\rho})\right|^2\right\}. \tag{28}$$

The speckle contribution to this quantity

$$A_T^{-1}\int d\overline{\rho}\,\mathcal{T}_f(\overline{\rho}) \tag{29}$$

is independent of $\overline{f}_i$ and $\overline{f}_r$. Thus, the speckle component of the multiplicative model is a random process which is uncorrelated between different aim points, and its contribution to the overall reflectance of the target is completely non-directional.

In a second analysis [12], the glint component $\mathbf{T}_{\mathrm{g}}(\overline{\rho})$ is ignored and the multiplicative model is expressly parameterized in terms of the position and orientation of the target

$$\mathbf{T}(\overline{\rho}) = \mathbf{T}_{\mathrm{s}}\left[\overline{\overline{\mathbf{U}}}_{\theta}(\overline{\rho}-\overline{\rho}_{\mathrm{c}})\right]\exp\left[\mathbf{j2k}\overline{\psi}^{\mathbf{T}}(\overline{\rho}-\overline{\rho}_{\mathrm{c}})\right], \tag{30}$$

where $\overline{\rho}_c$ is a target translation vector, $\overline{\overline{U}}_{\theta}$ is a matrix describing target rotation in the plane of illumination and $\overline{\psi}$ is a vector describing the tilt of the target with respect to the plane of illumination. Note that object tilt is equivalent to a change in aspect angle. The complex amplitude of the return due to illumination by a monochromatic beam of spatial pattern $\xi_{\mathbf{L}}(\overline{\rho})$ is given by

$$\mathbf{y} = \int \mathbf{T}(\overline{\rho})\xi_{\mathbf{L}}^{2}(\overline{\rho})\,\mathrm{d}\overline{\rho}. \tag{31}$$

The variation in target reflectivity is investigated by computing the correlation coefficient γ_{12} of the random variables $|\mathbf{y}_1|^2$ and $|\mathbf{y}_2|^2$, where the subscripts denote different choices of the parameters $\{\overline{\rho}_c, \overline{\overline{U}}_{\theta}, \overline{\psi}\}$. If $\mathbf{T}(\overline{\rho})$ is modeled as a complex Gaussian random process, then we have

$$\gamma_{12} = \frac{|E\{\mathbf{y}_1\mathbf{y}_2^*\}|^2}{E\{|\mathbf{y}_1|^2\}\,E\{|\mathbf{y}_2|^2\}}. \tag{32}$$

Speckle decorrelation is said to occur when the measurement parameters for the two different observations are such that $\gamma_{12} \leq e^{-2}$. For the specific case of object tilt $\overline{\psi}_1 - \overline{\psi}_2 = \Delta\overline{\psi}$, decorrelation occurs for

$$|\Delta\overline{\psi}| \geq \frac{2a}{\sqrt{4L^2 + k^2a^4}} \tag{33}$$

where a is the transmitted beam radius, L is the distance to the target and k is the wave number of the illumination. When only tilt is varied, the decorrelation limit is given approximately by $2/ka$ for near-field illumination and by a/L in the far-field. If multiple parameters are allowed to vary, the decorrelation limit is $2/ka$, regardless of the value of L.

Shapiro's model has yielded a measure of the expected variation in the reflectivity of a target for small changes in orientation, when the signal-target interaction can be modeled as taking place in a plane. In order to more directly relate this result to the case of an extended target on which all points are simultaneously illuminated by a wide radar beam, we will model the beam pattern $\xi_{\mathbf{L}}(\overline{\rho})$ with a constant value of ξ over the entire image plane. If we additionally ignore translation and in-plane rotation, then we have

$$E\{\mathbf{y}_1\mathbf{y}_2^*\} = \lambda^2\xi^4 \int d\overline{\rho}\,\mathcal{T}_{\int}(\overline{\rho})\rceil^{|\in\|(\overline{\psi}_\infty - \overline{\psi}_\in)^{\mathcal{T}}\overline{\rho}}. \tag{34}$$

2.0.1 The Proximity of Range Profile Surfaces

Ksienski and White [15, 6] investigate the nature of multi-frequency RCS measurements and their usefulness in target identification. Each vector of n measurements is modeled as the sum of an aspect-dependent, deterministic signal term and a noise term. Given that the signal part of a set of measurements for a particular aircraft depend only on a pair of angles (ϕ, θ), they reason that the set of all data points for each object must lie on a two-dimensional manifold in n-space. The problem of target identification then becomes one of assigning a vector of observations x to the data surface corresponding to one of the targets. For simple objects, the data surfaces for two different objects often occupy regions of n-space which are separable by hyper-planes. In this case, identification can be performed by linear discriminant; for any observation x the inner product with a weight vector w is computed and compared to a threshold T to determine on which side of the boundary hyper-plane the observation lies. For complex objects, two data surfaces, while still possibly occupying mutually exclusive regions of n-space, cannot in general be separated by hyper-planes. In this case identification is performed by finding the point on each of the data surfaces at a minimum distance in n-space from an observation x. The observation is classified as having been produced by the target corresponding to the nearer surface.

Given that one of these two identification algorithms is to be used, performance is to a great extent determined by the relative location of the data surfaces for each potential target. Specifically, do the surfaces for two targets intersect? If so, do intersections correspond to nearby aspect angles on the two surfaces? What is the proximity of the two surfaces where intersections do not occur? These questions are investigated by analysis of the data for two targets on a discrete grid in aspect angle. The behavior of the data surfaces in between grid points is approximated by bilinear interpolation. While it would be of extreme interest to know the sampling density in aspect angle at which bilinear interpolation would provide an accurate approximation to the actual data, such an analysis is not performed.

The first question addressed is that of intersection of two data surfaces. Four-frequency RCS measurements were simulated through computation for each of two targets (MIG-21 and F-4) on a rectangular grid in aspect angle. Bilinear interpolation is used to approximate surface behavior between grid points. An algorithm is presented which determines all intersections of the approximated surfaces. The authors correctly point out that regions in which the approximated surfaces intersect indicate regions in which the true surfaces must also intersect, but the converse is clearly not true. Thus, if the grid spacing in aspect angle is not small enough to accurately represent the data surface, intersections of the true surfaces may exist which are not indicated by the intersection of the approximated surfaces. The results of this analysis were that the approximated data surfaces did not intersect in the case of vertical polarization, but intersections were found in the case of horizontal polarization. Only one of these intersections corresponded to the same region in aspect angle for both targets. Thus, for aspects in this region, any algorithm would be expected to have difficulty in distinguishing between the MIG-21 and F-4 on the basis of the simulated four-frequency horizontal polarization data.

A second analysis addresses the question of the proximity of two data surfaces. This is computed as the minimum Euclidean distance of the two approximated surfaces over each rectangular region in aspect angle. As with the intersection study, the behavior of the true surfaces may be worse than that of the approximated surfaces. If the grid spacing is too large, the approximated surfaces fail to represent local deviations which may result in closer proximity. The study as performed compared four-frequency vertical polarization data simulated for the MIG-19 and F-104 aircraft. It was found that these two surfaces exhibited the greatest proximity to one another when the radar line-of-sight was within 15 degrees of the plane of the wings for both aircraft. Simulation studies were implemented to directly test the performance of a nearest neighbor classification algorithm on the simulated data. The highest rates of incorrect classification were found in the same aspect angle regions where the two data surfaces were most proximate.

3 Discussion of Open Issues

A comprehensive review of the open literature has revealed a plethora of algorithms for target identification, most of which rely on comparison of observations to stored templates for various targets and orientations. A number of models for HRR radar data have been described, and various authors have investigated the behavior of radar data and the implications for ATR. What questions have yet to be addressed?

It has been widely recognized in recent efforts [4, 5, 9, 1, 7] that identification of a dynamic aircraft is best performed by jointly estimating all parameters of interest: position, velocity, orientation, rotation rates, and target type. Position estimation or tracking has been studied extensively and a number of systems have been successfully implemented. Algorithms for identification, as reviewed above, have been proposed but generally require knowledge of orientation. The question of estimating the orientation of a target which produced a set of radar measurements has received less attention.

Many authors have noted that HRR radar data exhibit extreme variability for small changes in aspect angle, and some have attempted quantitative analyses of this variation. A debate has ensued as to the size of the library required to characterize HRR radar data and whether the size is so large as to preclude identification using these data. The analyses have been based solely on data sets taken for a particular target and a small number of orientations. Furthermore, these analyses have not focused on whether the variation of the data with orientation will allow for estimating the orientation which produced an observation. This could be investigated by computing the inner product or some appropriate distance metric between a given range profile and others resulting from the same target at nearby orientations.

As noted previously, Shapiro [12] has developed a model for laser radar reflectivity and used it to compute the orientation change which results in decorrelation of the return for a particular aim point. A model has not been found in the literature which reveals the expected variation of HRR data with target orientation. Such a model could be used to verify or reject statements made by other authors. An examination of the theory of random arrays suggests modeling a radar target as a collection of isotropic point reflectors randomly placed on a deterministic structure.

Ksienski and White make a bilinear approximation of the surface in range profile space corresponding to a single target at all possible orientations. Can this or another interpolation algorithm can be used to compute range profiles for orientations between library elements with acceptable accuracy? If so, the sampling density in orientation space required for this algorithm to be successful is of significant interest.

Ksienski and White make use of their bilinear representation to determine whether and where the data surfaces for two targets intersect, and to compute the proximity of these surfaces. Their analysis is applied to the case where the data is a series of four low frequency measurements of radar cross section. An application of similar techniques to range profiles would have much to say about the possibility of using these data in identification and the accuracy of prior knowledge of orientation necessary for successful identification.

Many authors have proposed algorithms for extracting target-specific information from radar data. Most have tested their algorithms using data which have been simulated by a computer program or collected from scaled models of targets in a compact radar range. What has apparently not been done is a direct analysis of HRR data taken from real aircraft. Can the range profiles provided by existing radar systems be used in target identification? Do range profiles collected from a particular aircraft more closely resemble others collected from the same aircraft than they do those collected from other aircraft at the same orientation? Note the connection to Ksienski's computation of the proximity of range profile surfaces. As a side issue, what types of preprocessing are necessary to remove extraneous dependence of HRR data on parameters other than orientation and target type (e.g. Doppler shift)?

4 Some Initial Results

4.1 Reflectivity Model Based on Random Arrays

4.1.1 Motivation

As mentioned previously, an integral part of any system which seeks to track and identify dynamic aircraft is the estimation of the orientation of the aircraft which produced a given set of radar measurements. This can be accomplished to within a certain accuracy, perhaps 10 degrees, when a history of position estimates and the aircraft dynamics are known. Successful identification, however, will generally require much more accurate estimates of orientation.

The question as to whether HRR radar data can be used in orientation estimation remains open. The work of Shapiro has provided a model for illumination of a planar target by a laser radar and an estimate of the angular change over which the return will decorrelate. However, no model for radar reflectivity has been used to predict the variation of HRR radar data with the orientation of the target. This section proposes such a model.

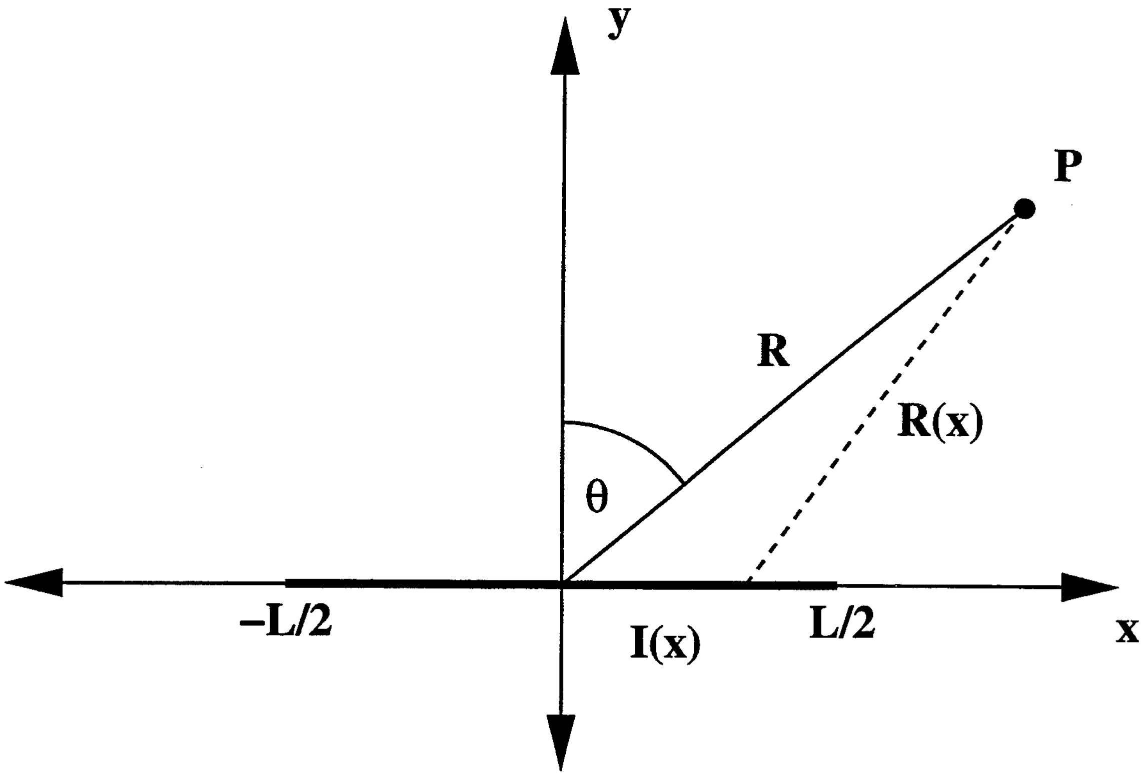


Figure 1: Linear radiator lying in the plane.

The model developed here draws heavily upon the theory of random arrays, and relies on the assumption that the following measurement paradigms are essentially equivalent.

- A linear array of isotropic receivers is exposed to the signal produced by a single distant point source. The output signal is the superposition of the signals received by each of the array elements.
- A linear array of isotropic point sources is observed by a single distant receiver. The output signal is the output of the receiver.
- A linear array of isotropic point reflectors is illuminated by a monostatic radar system. The output signal is the reflected signal observed by the radar.

These scenarios are equivalent in the sense that the same mathematical model can be used to predict the behavior of the output signal as the orientation of the array is varied. The development of this model follows the treatment of random arrays by Steinberg [14], with additional insight drawn from Goodman [2] and Johnson [3].

4.1.2 The Linear Radiator

Figure 1 displays a radiating element of length L lying on the x-axis in the plane. Each point x on the radiator is modeled as an independent source of monochromatic radiation with wave number $k = 2\pi/\lambda$ and intensity $I(x)$. According to the Huygens-Fresnel principle, the scalar field at any point P is given by the linear superposition of fields resulting from all of the sources

$$E(P) = \int_{-\frac{L}{2}}^{\frac{L}{2}} I(x) \frac{e^{-jkR(x)}}{R(x)} dx, \tag{35}$$

where $R(x)$ is the distance from the observation point P to a point on the radiator

$$R(x) = \sqrt{(x - p_x)^2 + p_y^2}, \tag{36}$$

and the effects of propagation are modeled as those associated with an expanding spherical wave in free space.

Simplifications of this model are achieved through approximations based on the assumption that the distance $R(x)$ is much larger than the radiator dimension L. If we define $R = \sqrt{p_x^2 + p_y^2}$ as the distance from point P to the origin, then an exact expression for $R(x)$ is given by

$$R(x) = R\sqrt{1 - \frac{2xp_x}{R^2} + \frac{x^2}{R^2}}. \tag{37}$$

Approximations to this expression are made by retaining a sufficient number of terms in the binomial series expansion for the square root. In the denominator, only the constant term is retained

$$R(x) \approx R, \tag{38}$$

and in the complex exponential, the constant and linear terms are retained

$$R(x) \approx R\left(1 - \frac{xp_x}{R^2} + \frac{x^2}{R^2}\right). \tag{39}$$

This leads to the following expression for the observed field

$$E(P) = \frac{e^{-jkR}}{R}\int_{-\frac{L}{2}}^{\frac{L}{2}} I(x)\exp\left[-jk\left(-\frac{xp_x}{R} + \frac{x^2}{2R}\right)\right]dx. \tag{40}$$

Further simplification of this expression is achieved by making the more restrictive far-field or Fraunhofer assumption, which requires that the distance R be very large compared to the ratio of the square of the radiator dimension to the wavelength of the radiation

$$R \gg \frac{\pi L^2}{\lambda}. \tag{41}$$

In this case the quadratic phase term is dropped. If we also define

$$u = \cos\theta = \frac{p_x}{R}, \tag{42}$$

then we have

$$E(R,u) = \frac{e^{-jkR}}{R}\int_{-\frac{L}{2}}^{\frac{L}{2}} I(x)e^{jkxu}dx. \tag{43}$$

Finally, $E(R,u)$ and $I(x)$ are replaced by normalized quantities,

$$f(u) \triangleq \frac{E(R,u)}{E(R,u=0)} \tag{44}$$

$$i(x) \triangleq \frac{I(x)}{\int_{-\frac{L}{2}}^{\frac{L}{2}} I(x)dx}, \tag{45}$$

which yields

$$f(u) = \int_{-\frac{L}{2}}^{\frac{L}{2}} i(x)e^{jkxu}dx. \tag{46}$$

Note that the above expression suppresses any dependence on R; the radiation pattern for points in the far-field is expressed solely as a function of the direction relative to the radiator through the variable u. Note also that if $i(x)$ is defined to be identically zero outside the extent of the radiator, then this quantity and the radiation pattern $f(u)$ are related through the spatial Fourier transform in one dimension.

As an example, consider the case where the intensity $I(x)$ is constant over the extent of the radiator,

$$I(x) = I \Longleftrightarrow i(x) = 1 \qquad \forall x \in \left[-\tfrac{L}{2}, \tfrac{L}{2}\right]. \tag{47}$$

The radiation pattern in this case is simply a sinc function in the variable u

$$f(u) = \frac{\sin(kuL/2)}{kuL/2}. \tag{48}$$

Note that, because $u = \cos\theta \in [-1, 1]$, the zeros associated with the sinc function will be seen in $f(u)$ only when $L > \lambda$.

4.1.3 Linear Arrays

Now consider replacement of the continuous radiator with a linear array consisting of a finite number N of isotropic sources of monochromatic radiation at discrete positions

$$x_n \in \left[-\frac{L}{2}, \frac{L}{2}\right] \qquad n = 1, \ldots, N. \tag{49}$$

The normalized source intensity is given by

$$i(x) = \sum_{n=1}^{N} i_n \delta(x - x_n), \tag{50}$$

where the individual intensities i_n are assumed to sum to unity, and the far-field radiation pattern is given by

$$f(u) = \sum_{n=1}^{N} i_n e^{jkx_n u}. \tag{51}$$

In the case of uniform element spacing and uniform intensity, the radiation pattern reduces to

$$f(u) = \sum_{m=-\infty}^{\infty} \frac{\sin\left[\frac{kL}{2}\left(u - \frac{m\lambda}{d}\right)\right]}{\frac{kL}{2}\left(u - \frac{m\lambda}{d}\right)}, \tag{52}$$

where $d = \frac{L}{N-1}$ is the element spacing.

Now consider a linear array of elements which produce monochromatic radiation of equal intensity, and whose positions within the array x_n are independent identically distributed random variables with probability density function $w_{\mathrm{x}}(x)$. In this case the far-field radiation pattern is a random process, the mean of which is given by the characteristic function of the random variables x_n.

$$\begin{aligned} E\{f(u)\} &= \frac{1}{N}\sum_{n=1}^{N} E\left\{e^{jkx_n u}\right\} && (53) \\ &= \phi_{\mathrm{x}}(ku). && (54) \end{aligned}$$

The covariance function can be found through similar analysis

$$\begin{aligned} E\{f(u_1)f^*(u_2)\} &= \frac{1}{N^2}\sum_{n=1}^{N}\sum_{m=1}^{N} E\left\{e^{jk(x_n u_1 - x_m u_2)}\right\} && (55) \\ &= \frac{1}{N}\left[\phi_{\mathrm{x}}(ku_1 - ku_2) + (N-1)\phi_{\mathrm{x}}(ku_1)\phi_{\mathrm{x}}(-ku_2)\right]. && (56) \end{aligned}$$

As an example, consider a probability density function which is constant over a linear array of length L, and zero elsewhere.

$$w_{\mathrm{x}}(x) = \begin{cases} L^{-1} & x \in [-\frac{L}{2}, \frac{L}{2}] \\ 0 & \text{otherwise} \end{cases} \tag{57}$$

The associated characteristic function is given by the Fourier transform of $w_{\mathrm{x}}(x)$.

$$\phi_{\mathrm{x}}(\omega) = \frac{\sin(\omega L/2)}{\omega L/2} \tag{58}$$

Thus, the radiation pattern associated with this random array has an expected value given by

$$E\{f(u)\} = \frac{\sin(kuL/2)}{kuL/2}, \tag{59}$$

which is identical to the radiation pattern of the linear radiator of constant intensity.

To see that the assumption that radiating elements are of uniform intensity is not restrictive, consider a linear array of elements with random positions x_n, and with normalized intensities i_n that are modeled as unknown deterministic parameters which sum to unity. In this case, the expected radiation pattern is unchanged:

$$\begin{aligned} E\{f(u)\} &= \sum_{n=1}^{N} E\left\{i_n e^{jkx_n u}\right\} && (60) \\ &= \phi_{\mathrm{x}}(ku) \sum_{n=1}^{N} i_n && (61) \\ &= \phi_{\mathrm{x}}(ku). && (62) \end{aligned}$$

The expected behavior of a random array is therefore determined by the spatial distribution of the elements. The expected radiation pattern of a random array with elements of uniform intensity and positions distributed according to $w_{\mathrm{x}}(x)$ will be equal to the deterministic angular pattern of the linear radiator with intensity profile $i(x)$ whenever

$$w_{\mathrm{x}}(x) = i(x). \tag{63}$$

4.1.4 Reflectivity Model for a square patch

The results of the previous section are generalized to consider a two-dimensional radiator observed from a point in three-dimensional space (see figure 2). A contiguous region in the (x, y) plane surrounding the origin is filled with isotropic sources of monochromatic radiation with normalized intensity function $i(x, y)$. The normalized scalar field observed at a point in the far-field at azimuth angle ϕ and elevation angle θ is given by the two-dimensional spatial Fourier transform of the normalized intensity

$$f(u, v) = \int\int i(x, y) e^{jk(xu+yv)}\, dx\, dy, \tag{64}$$

where $u = \sin\theta\cos\phi$ and $v = \sin\theta\sin\phi$. In the above expression, integration is performed over the entire plane and $i(x, y)$ is equal to zero outside the extent of the radiator.

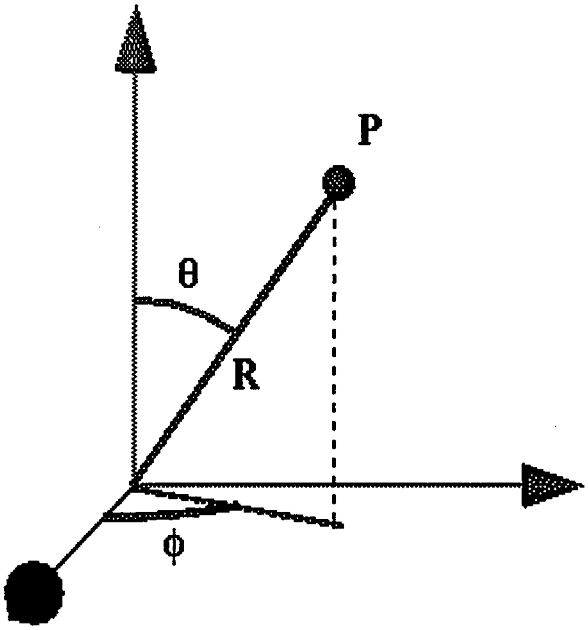


Figure 2: Geometry for determining radiation pattern of planar and volume radiators.

A planar array of N isotropic sources with positions (x_n, y_n) and intensities i_n has normalized intensity

$$i(x,y) = \sum_{n=1}^{N} i_n \delta(x - x_n)\delta(y - y_n) \tag{65}$$

and far-field radiation pattern

$$f(u,v) = \sum_{n=1}^{N} i_n e^{jk(x_n u + y_n v)}. \tag{66}$$

The elemental positions are modeled as i.i.d. random variables with probability density function $w_{\mathrm{xy}}(x,y)$, and the intensity is assumed constant over the array. The expected value of the radiation pattern is again given by the characteristic function associated with the random positions

$$\begin{aligned} E\{f(u,v)\} &= \frac{1}{N}\sum_{n=1}^{N} E\left\{e^{jk(x_n u + y_n v)}\right\} &(67)\\ &= \phi_{\mathrm{xy}}(ku, kv). &(68) \end{aligned}$$

Let us apply this model to the example of a flat square patch. Isotropic radiators are uniformly distributed over a square of dimension L in the (x,y) plane

$$w_{\mathrm{xy}}(x,y) = \begin{cases} L^{-2} & (x,y) \in [-\frac{L}{2}, \frac{L}{2}]^2 \\ 0 & \text{otherwise.} \end{cases} \tag{69}$$

$$\phi_{\mathrm{xy}}(\omega, \nu) = \frac{\sin(\omega L/2)}{\omega L/2}\frac{\sin(\nu L/2)}{\nu L/2}. \tag{70}$$

The normalized far-field radiation pattern is a random process with mean and covariance given by

$$E\{f(u,v)\} = \frac{\sin(kuL/2)}{kuL/2}\frac{\sin(kvL/2)}{kvL/2} \tag{71}$$

$$\begin{aligned} &E\{f(u_1,v_1)f^*(u_2,v_2)\} = \\ &\quad \frac{1}{N}\left[\frac{\sin\left(k(u_1-u_2)L/2\right)}{k(u_1-u_2)L/2}\frac{\sin\left(k(v_1-v_2)L/2\right)}{k(v_1-v_2)L/2} + \right. \\ &\quad \left. (N-1)\frac{\sin(ku_1L/2)}{ku_1L/2}\frac{\sin(kv_1L/2)}{kv_1L/2}\frac{\sin(ku_2L/2)}{ku_2L/2}\frac{\sin(kv_2L/2)}{kv_2L/2}\right]. \end{aligned} \tag{72}$$

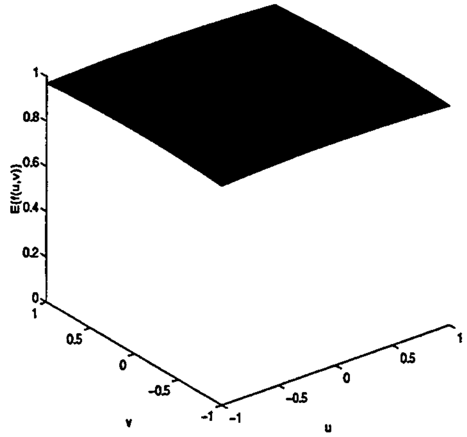


Figure 3: Expected radiation pattern of a square patch with randomly placed radiators. The patch dimension $L = \lambda/10$.

Insight can be gained by examining the expected radiation pattern for different ratios of the patch dimension L to the radiation wavelength λ. For $L < \lambda$, the zeros of $f(u, v)$ occur outside $[-1, 1]^2$, and are therefore never reached for any angles (ϕ, θ). Figure 3 is a plot of the expected radiation pattern for $L = \lambda/10$. This figure demonstrates that the random array model prescribes that targets significantly smaller then the illumination wavelength will appear to an observer in the far-field essentially as point sources whose return is independent of orientation.

For $L = \lambda$, the first zeros lie on the border of the visible region, $|u| = 1$ or $|v| = 1$, as seen in figure 4. This region corresponds to eight points in azimuth and elevation

$$(\theta, \phi) \in \left\{ (k\frac{\pi}{2}, l\frac{\pi}{2}) : k = \pm 1, l = 0, 1, 2, 3 \right\}. \tag{73}$$

Hence, whenever the patch is observed from a point in the far-field on either the x- or the y-axis, the patch "disappears" in the sense that the scalar field at the observer has a mean value of zero. The reason for this effect is more easily understood by considering the deterministic square radiator. When observed from the x- or y-axis, elemental radiators are spread over exactly one wavelength in range, resulting in complete phase cancellation in the observed field.

When $L > \lambda$, the expected radiation pattern becomes highly directional. Figure 5 shows the result for $L = 10\lambda$. The peak response is reached for $u = v = 0$. This corresponds in azimuth to $\theta \in \{0, \pm\pi\}$, when the plane of the patch is perpendicular to the observation vector.

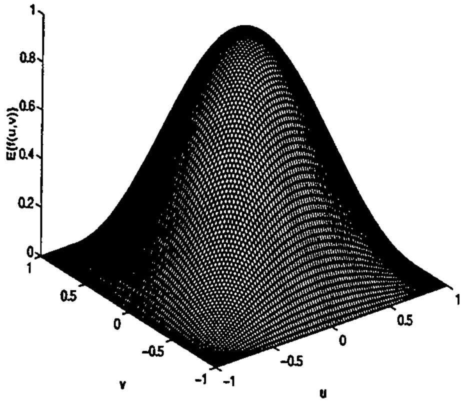


Figure 4: Expected radiation pattern of a square patch with randomly placed radiators. The patch dimension $L = \lambda$.

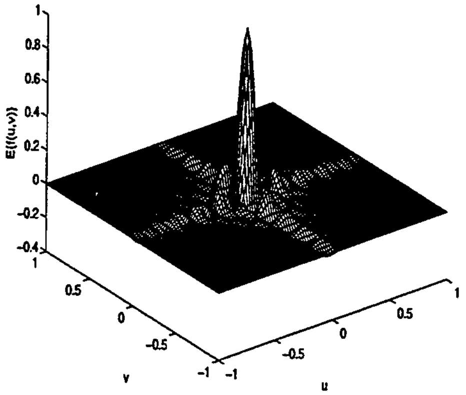


Figure 5: Expected radiation pattern of a square patch with randomly placed radiators. The patch dimension $L = 10\lambda$.

In this section, we have developed a model for the reflectivity of a square planar patch based on the theory of random arrays. This model allows us to compute the mean, variance and other moments of the radiation pattern. Analysis of the resulting expressions yields results which are both intuitive and in agreement with other models for this phenomenon.

4.1.5 Extending the Model

The model so far developed has allowed us to generate a statistical characterization of the scalar field resulting from random placement of identical isotropic sources of monochromatic radiation onto simple deterministic structures. The combined effect of the far-field assumption, which expresses the observed field in terms of a Fourier integral on the distribution of point sources, and the use of simple geometric shapes, which allows for evaluation of the integral, has yielded analytic expressions for the moments of the radiation pattern. Based on the equivalence discussed at the beginning of this section, we expect this behavior to translate well to the case of a collection of randomly placed isotropic point scatterers illuminated by a monostatic radar system. Ultimately, we seek a model which will predict the behavior of multidimensional HRR returns from complex 3-D objects as their orientation is varied. The generalizations necessary to achieve this goal are described below.

Three-Dimensional Arrays The case illustrated in fig. 2 is generalized to allow the radiator to have extent in the z dimension. The scalar field at point P is given by

$$E(P) = \int\int\int I(x,y,z)\frac{e^{-jkR(x,y,z)}}{R(x,y,z)}dx\,dy\,dz, \tag{74}$$

and the distance from P to a point in the array is

$$\begin{aligned} R(x,y,z) &= \sqrt{(x-p_x)^2+(y-p_y)^2+(z-p_z)^2} && (75)\\ p_x &= R\sin\theta\cos\phi && (76)\\ p_y &= R\sin\theta\sin\phi && (77)\\ p_z &= R\cos\theta. && (78)\end{aligned}$$

Under the far-field assumption, this reduces to

$$\begin{aligned} E(P) = \frac{e^{-jkR}}{R}\int\int\int I(x,y,z)\times \\ \exp\left[jk(x\sin\theta\cos\phi + y\sin\theta\sin\phi + z\cos\theta)\right]dx\,dy\,dz. \end{aligned} \tag{79}$$

Note that for a planar array, $z \equiv 0$ and, after normalization, the above expression reduces to the 2-D spatial Fourier transform relation given previously. With u and v as defined previously and $w = \cos\theta$, the field at point P can be expressed as a spatial Fourier transform in three dimensions. After normalization, we have

$$f(u,v,w) = \int\int\int i(x,y,z)e^{jk(xu+yv+zw)}dxdydz. \tag{80}$$

A volume array of N isotropic sources at positions (x_n, y_n, z_n) and with intensities i_n has its far-field radiation pattern given by

$$f(u,v,w) = \sum_{n=1}^{N} i_n e^{jk(x_n u + y_n v + z_n w)}. \tag{81}$$

Our development to this point has investigated the radiation pattern resulting from a collection of isotropic sources. The reason for doing so has been that this scenario is essentially equivalent to the case of illumination by a single source of a collection of isotropic reflectors, and hence $f(u,v,w)$ is a good model for the reflectance pattern of a target illuminated by a monostatic radar system. An important difference, however, is that the path length from source to sensor in the case of monostatic radar is twice that found in the case of a remotely sensed array of sources. Thus, the change in path length induced by changing the orientation of the target, and therefore the change in the phase of the received signal, is likewise doubled. The proper expression, then, for the reflectance pattern resulting from illumination of a volume array of isotropic reflectors at positions (x_n, y_n, z_n) by a distance source of electromagnetic radiation of wave number k is

$$f(u,v,w) = \sum_{n=1}^{N} i_n e^{j2k(x_n u + y_n v + z_n w)}. \tag{82}$$

A volume random array of isotropic reflectors with i.i.d. elemental positions distributed according to $w_{\mathrm{xyz}}(x,y,z)$ has a normalized reflection pattern with mean

$$\begin{aligned} E\{f(u,v,w)\} &= \frac{1}{N}\sum_{n=1}^{N} E\left\{e^{j2k(x_n u + y_n v + z_n w)}\right\} && (83)\\ &= \phi_{\mathrm{xyz}}(2ku, 2kv, 2kw), && (84) \end{aligned}$$

and correlation

$$\begin{aligned} & E\{f(u_1,v_1,w_1)f^*(u_2,v_2,w_2)\} \\ &= \frac{1}{N^2}\sum_{n=1}^{N}\sum_{m=1}^{N} E\left\{e^{j2k[(x_n u_1 - x_m u_2)+(y_n v_1 - y_m v_2)+(z_n w_1 - z_m w_2)]}\right\} && (85)\\ &= \frac{1}{N}\left[\phi_{\mathrm{xyz}}(2ku_1 - 2ku_2, 2kv_1 - 2kv_2, 2kw_1 - 2kw_2)\right.\\ &\quad \left. + (N-1)\phi_{\mathrm{xyz}}(2ku_1, 2kv_1, 2kw_1)\phi_{\mathrm{xyz}}(-2ku_2, -2kv_2, -2kw_2)\right]. && (86) \end{aligned}$$

Note that, due the the dependence of (u, v, w) on the orientation angles (θ, ϕ), the domain of the radiation pattern $f(u, v, w)$ is restricted to the unit sphere. The implications of this restriction will be discussed further below.

Application to HRR Radar Data In the radar context, our development thus far allows for computation of the expected variation with orientation of the complete return from the target when illuminated by a signal of constant wavelength. This quantity is therefore closely related to the single-frequency radar cross section. While the variation in RCS with orientation is an interesting subject which has been studied through experimentation and simulation by a number of authors, our primary interest lies in how the HRR range profile will change with orientation. Ultimately, we would desire a model which provides for each target a function of the form

$$g(R, \theta, \phi) \tag{87}$$

which, for each aspect angle pair (θ, ϕ), describes the reflectivity of the target as a function of range. If this quantity is modeled as a random process, then examination of the behavior of moments such as

$$E\left\{\int g(R; \theta_1, \phi_1) g^*(R; \theta_2, \phi_2)\, dR\right\} \tag{88}$$

will be indicative of the performance of any algorithm which seeks to estimate orientation using such data.

In practice, measurement or simulation of the range profile of a given target is performed by measuring or simulating an appropriate function of time. This is due to an assumed equivalence between the range of a given point on the target and the two-way delay introduced to a signal reflecting off of that point. In any case the data which is observed from a target as well as the library of templates to which observations will be compared in performing orientation estimation will be samples of time functions. Thus, we are interested in moments such as

$$\gamma_{12} = E\left\{\int g(t; \theta_1, \phi_1) g^*(t; \theta_2, \phi_2)\, dt\right\} \tag{89}$$

The straightforward method for obtaining a range profile of a radar target is to transmit a pulse of sufficiently short duration to provide the necessary range resolution, and examine the return in the time domain. Alternatively, one may transmit a signal of sufficiently wide bandwidth. This is done either by making a series of single frequency measurements or by transmitting a chirp pulse, in which the frequency is swept linearly over the appropriate range, and sampling the return in time. The range profile is often obtained from this frequency domain data directly through the inverse Fourier transform.

Applying this method to the random array model, we begin by defining a reflection pattern for which the wave number of the source is allowed to vary.

$$f(ku, kv, kw) \tag{90}$$

The expressions describing the moments of this reflection pattern for a volume random array are unchanged; the change in notation serves only to indicate that the wavelength of the source is no longer fixed. For any k, evaluating this expression at different orientations corresponds to evaluation at different points on a sphere of radius k. By allowing k to vary, we can find the value of this function at any point in the 3-D Fourier domain of f.

The range profile is defined as the inverse Fourier transform of this quantity,

$$g(t;\theta,\phi) = \frac{1}{2\pi}\int f(ku,kv,kw)e^{j\omega t}d\omega \tag{91}$$

where $k = \omega/c$. Invoking Parseval's relation, we have

$$\begin{aligned}
&\gamma_{12} \\
&= E\left\{c\int f(ku_1,kv_1,kw_1)f^*(ku_2,kv_2,kw_2)\,dk\right\} &(92)\\
&= \frac{c}{N}\int [\phi_{\mathrm{xyz}}(2ku_1-2ku_2,2kv_1-2kv_2,2kw_1-2kw_2) \\
&\quad + (N-1)\phi_{\mathrm{xyz}}(2ku_1,2kv_1,2kw_1)\phi_{\mathrm{xyz}}(-2ku_2,-2kv_2,-2kz_2)]\;dk. &(93)
\end{aligned}$$

Complex 3-D Targets Either the exact or approximate expressions given above will allow evaluation of the radiation pattern for a three-dimensional radiator or array via a Fourier integral. However, our ability to obtain analytic expressions for radiation patterns has been strictly dependent on the choice of linear and square planar arrays. For example, evaluation of the radiation pattern for a triangular patch in the (x,y) plane quickly becomes mathematically cumbersome as one tries to properly define the limits of integration. Similar analytic evaluation for an array which is distributed over the surface of a complex three-dimensional object would be exceedingly intractable. It is not clear at this time how this issue can be resolved.

Relationship to Shapiro's Model As discussed previously, Shapiro has presented a model for the response of a planar target to illumination by a laser radar and has analyzed the model to consider the change in orientation required to produce decorrelation in the response. We have slightly extended his analysis to consider the illumination of the target by a wide beam, and produced an expression for the correlation between responses for two different orientations. Shapiro's model is based on describing the reflectivity of the target as a spatial random process, the domain on which is the plane in which reflection takes place. The change in the reflectivity with orientation is ascribed to the change in the phase of reflections from points on the target due to an adjustment of the range of these points. The model developed in this section has arrived at a similar result from different assumptions; here the responses from individual reflectors are deterministic and the random nature and spatial dependence of the reflectivity arise out of random placement of reflectors on the surface of the target. Like Shapiro, the effects of changing orientation are due to the change in the phase of the return from individual reflectors. Unification of these models may be achieved by allowing the intensity profile $i(x,y,z)$ in the random array model to also be a random process. In any case, it is clear that quantitative comparison of these two models to each other and to known results would provide an important test of their validity.

4.2 Simulation Results

This section discusses three studies that have been made concerning the variability of simulated range profiles for small changes in orientation. Returns from two simple targets were simulated using xpatch over different regions in orientation space. Most algorithms for orientation estimation and identification compare observations with a library of stored templates. We examine the behavior of the Euclidean distance between a simulated range profile and those drawn from nearby orientations, to see if the data could successfully be used in orientation estimation via either a global search or a gradient-based local search.

4.2.1 Aim-92 Missiles

In our first study, xpatch was used to simulate range profiles for a wireframe model of a pair of Aim-92 missiles. The missiles are each approximately 150 inches long and 40 inches in diameter and are in a fixed side-by-side position relative to each other. The return is computed for each of 1024 bins covering 200 inches in range, which included the entire extent of the target. The elevation angle was fixed at -15 degrees while the azimuth was varied between -15 and -25 degrees; the sampling in azimuth was nonuniform, with the sampling density increasing in the vicinity of -20 degrees. This nonuniform sampling was chosen to allow for examination of the behavior of the range profile on multiple scales in azimuth while keeping the total number of computed range profiles reasonable.

Suppose we are given the range profile at -20 degrees azimuth as an observation, and the entire set of range profiles as a library. Can we use these data to estimate the azimuth which resulted in the observation? Figure 6 is intended to address this question. On the left, the Euclidean distance between each range profile and the range profile at -20 degrees azimuth is plotted. It is clear that for the case examined here, orientation estimation is indeed possible; the distance between the observation and the range profile corresponding to the true orientation (identically zero) is much different from the computed distance for any other azimuth. Clearly this sharp spike could be located with a global search over an appropriate interval in azimuth.

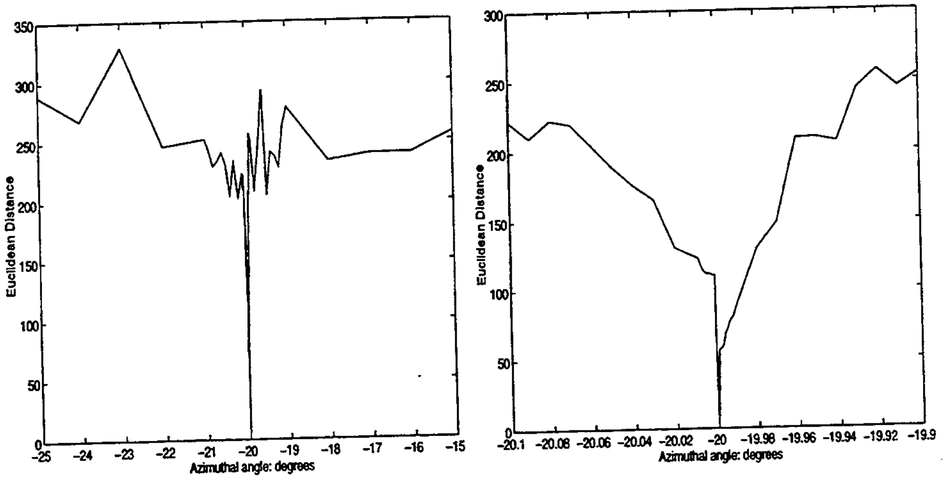


Figure 6: Euclidean distance between range profiles of a pair of missiles. Range profiles have been simulated for a sampling in azimuth angles between −15 and −25 degrees. Distance between each range profile and the range profile at −20 degrees is plotted. The plots show that orientation estimation from these data is possible using either a global or a gradient-based search, given sufficiently fine sampling of orientation space and an accurate initial guess.

The plot on the right of figure 6 is an enlargement showing the behavior in the vicinity of −20 degrees. Both plots demonstrate the extreme variability of the simulated range profiles for small changes in orientation. The width of the main lobe indicates that sampling finer than 0.1 degrees in azimuth is necessary to characterize the range profile. Furthermore, the enlargement shows that orientation estimation via a gradient search is possible using these data, provided that an initial guess of the true orientation is available which is in error by not more than 0.1 degrees. The slope of this graph is generally negative to the left of −20 degrees and positive to the right. Thus the local behavior of this metric would quickly push a gradient based search to the global minimum at the truth.

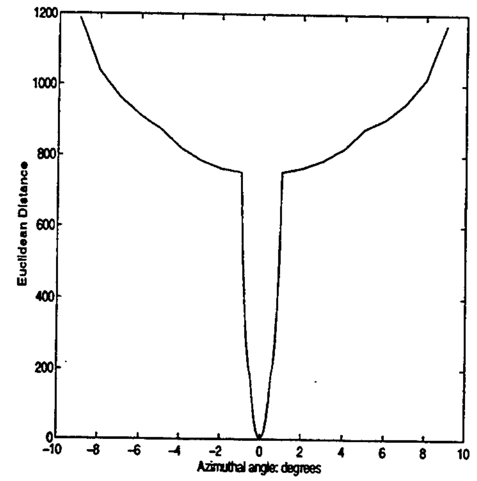


Figure 7: Euclidean distance between range profiles for box target. Range profiles have been simulated for a sampling of azimuth angles between ±9 degrees. Distance between each range profile and the range profile for 0 degrees is plotted. The smooth variation in distance with azimuth will allow for orientation estimation using a gradient search even when the initial estimate has significant error.

4.2.2 Rectangular Box

To see whether the behavior previously observed is exhibited by other targets, a similar study was performed using a rectangular box. The box dimensions are 32 × 34 × 36 inches. Range profiles were simulated for 0 degrees elevation and for azimuth angles between ±9 degrees. The Euclidean distance between each range profile and the range profile at 0 degrees azimuth is plotted in figure 7. The wide main lobe indicates that sampling in azimuth on the order of 1 degree may be sufficient to represent the data. Additionally, the extremely smooth nature of this graph implies that gradient-based orientation estimation is possible, even when the initial guess is in error by several degrees.

One might wonder whether the extremely favorable behavior seen in the previous study was a function of the simplicity of the target. A third study has shown that this is not the case. This study is identical to the second, except that the box has been rotated so that the nominal azimuth angle is 45 degrees.

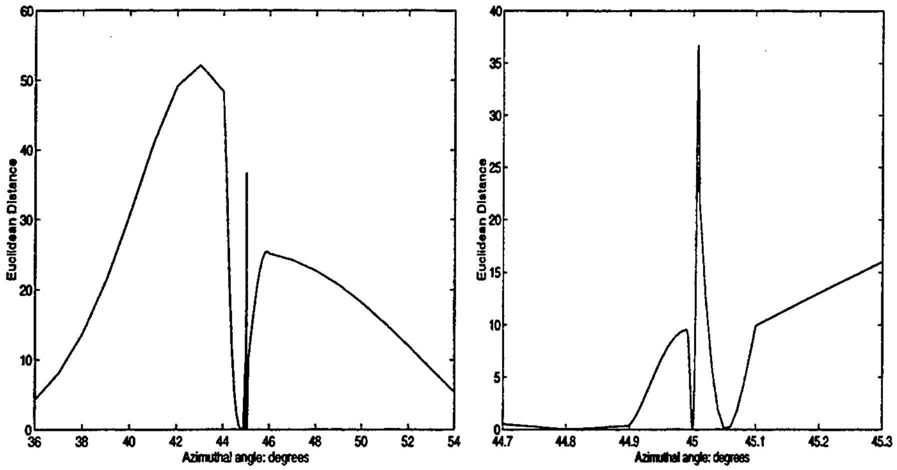


Figure 8: Euclidean distance between range profiles for box target. Range profiles have been simulated for a sampling of azimuth angles between 36 and 54 degrees. Distance between each range profile and the range profile for 45 degrees is plotted. The behavior exhibited here indicates that orientation estimation using these data would be difficult with either a global or a gradient-based search.

In figure 8 the left plot shows the Euclidean distance between the observation and range profiles computed for azimuths between 36 and 54 degrees, and the right plot is an enlargement in the vicinity of 45 degrees. These plots show that orientation estimation using a global search will likely result in errors, as there are many azimuths near the true value for which the distance between the associated range profile and the observation is nearly zero. Furthermore, even if an initial guess of azimuth is available which is accurate to within 0.1 degrees, the slope of the graph would tend to push a gradient based search away from the truth and toward a local minimum.

We have conducted three studies of the variation of the range profile with orientation, using data simulated for different targets and nominal orientations. The results vary widely in their implications for the sampling density in orientation required to describe the behavior of the range profile, and for the usefulness of these data in estimating the orientation corresponding to an observed range profile using either a global or gradient-based search. It is clear that the behaviors studied are highly dependent on both the target type and the nominal orientation.

4.3 Analysis of Real HRR Data

In this section, the results of analysis of range profiles collected by an actual radar system from aircraft in flight are presented. The collection of the range profiles was performed several years prior to this analysis. The only sources of information regarding the conditions of the experiment and the nature of the data are a few pages of handwritten notes, which are incomplete, and the data encoded in the header of each range profile, which are frequently uninformative or self-inconsistent. As such, it has been necessary to make a number of assumptions regarding the measurement scenario and the nature of corruptions of the data. These assumptions have been based on any actual knowledge available, examination of the data in the time and frequency domains, and common sense where no direct evidence is available.

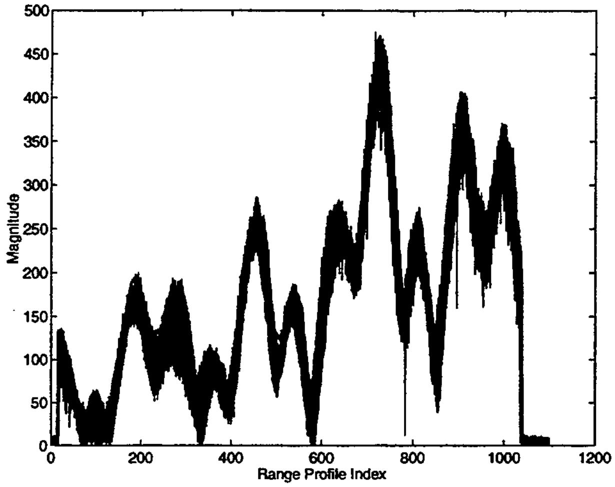


Figure 9: Magnitude of 119 range profiles observed for the Cessna 310 aircraft. The high degree of similarity for all the range profiles in this data set supports the model of the range profile as the sum of a deterministic, orientation-dependent signal and a random noise process.

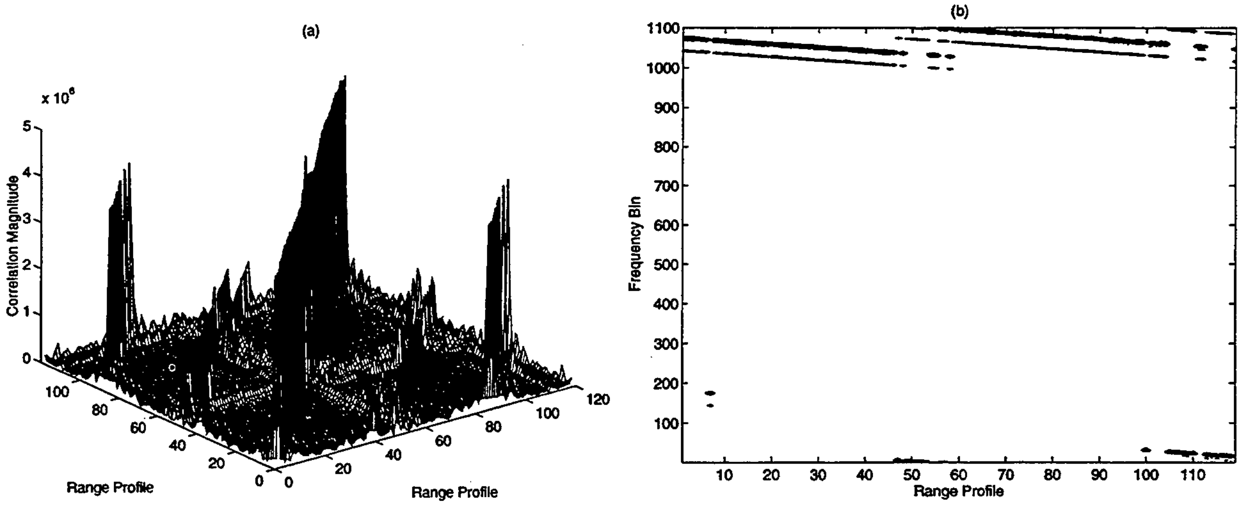


Figure 10: Effects of variation in the phase of range profiles from the DC-10 data set. The absolute inner product between pairs of range profiles is shown in (a). The magnitude of the fast Fourier transform of each range profile is shown in (b). Note that peaks in correlation correspond to pairs of range profiles that occupy the same frequency band.

We have been provided with four sets of range profiles for each of six different civilian aircraft. These data were collected in an experiment performed by Rome Laboratory in which targets of opportunity were illuminated by an S-band radar. Vertical polarization was used during transmission and reception of radar signals. Each of the 24 data sets consists of 119 range profiles collected over a time interval of less than one second, with successive range profiles occurring approximately 5 milliseconds apart. The change in the orientation of the illuminated aircraft with respect to the radar system between adjacent range profiles is exceedingly small. Therefore, if the range profile can be correctly modeled as the sum of a deterministic, orientation-dependent signal and a random noise process, then one would expect that the range profiles in a given data set for a particular aircraft would be very similar to one another.

Figure 9 shows that this is indeed the case for one of the data sets. In this figure, plots of the magnitude of each of 119 range profiles of the Cessna 310 are superimposed. The self-consistency of the range profiles in this data set is evident in that the individual traces which make up this plot lie nearly on top of one another, leading to the conjecture that these range profiles should be highly correlated with one another. This turns out not to be the case, as the phases of the individual range profiles vary significantly from one to the next.

The mechanism responsible for the phase variation is illustrated in figure 10. One set of range profiles for the DC-10 aircraft was selected. The magnitude of the inner product computed between pairs of range profiles from this set is shown in 10(a). Thus the main diagonal in this figure indicates the energy in each range profile. What is striking about this figure is that the highest peaks other than the main diagonal occur some 90 range profiles away, indicating that a given range profile is much more highly correlated with a range profile collected nearly half a second later than it is with any of those in between. The reason for this is shown in 10(b), which is a contour plot of the magnitude of the fast Fourier transform of each of the range profiles. Note that the frequency domain representation of the range profiles are very similar except that they are shifted relative to one another; peaks in correlation correspond to pairs of range profiles that occupy roughly the same frequency band.

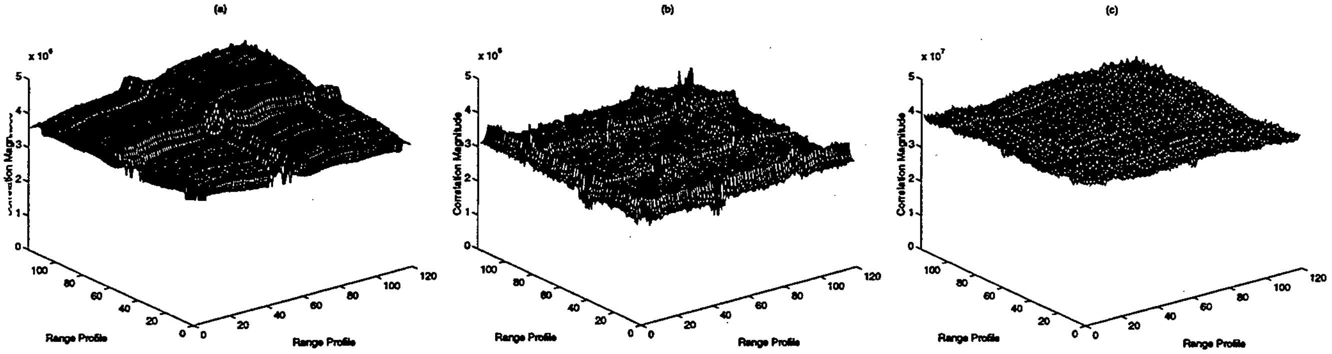


Figure 11: Absolute inner product of pairs of range profiles after applying frequency shift. (a) DC-10 data set. (b) DC-9 data set. (c) Cessna 310 data set. The flatness of these plots indicates the similarity of the range profiles within each set. Note that the energy of the range profiles varies between data sets by an order of magnitude. This will be eliminated through normalization.

There are two distinct frequency shifts: one varies linearly with the range profile index and the other consists of discontinuous jumps. A possible explanation is that the linear component is a differential Doppler shift induced by an approximately linear acceleration of the aircraft relative to the radar, and that the discontinuities are the result of an attempt to correct for the Doppler shift based on an estimate of the target velocity over each illumination interval. In any case, the general appearance of this frequency plot is replicated over all 24 sets of range profiles, indicating that the range profiles in a given set are self-consistent, except for a relative frequency shift.

On the assumption that the observed frequency shifts are erroneous, an attempt was made to correct for them by multiplying each range profile by a complex exponential such that the peak of the frequency response was shifted to DC. Figure 11 illustrates the results of this correction. The absolute inner product of pairs of corrected DC-10 range profiles is shown in 11(a), and should be compared with 10 (a). Note that the correlation between corrected range profiles is high across the entire data set. Similar frequency shifts were applied to range profiles collected for the DC-9 and Cessna 310 aircraft, and the resulting pairwise inner products are plotted in 11(b) and 11 (c), respectively. The fact that individual range profiles are highly correlated with others observed for the same aircraft indicates that the process of range profile measurement is highly repeatable, that is, the effects of receiver noise and other distortions introduced in the measurement process are not severe enough to drastically change the observed range profile. In particular, the change in orientation relative to the radar system experienced by these aircraft during an illumination interval on the order of one second is small enough that the range profile remains relatively constant.

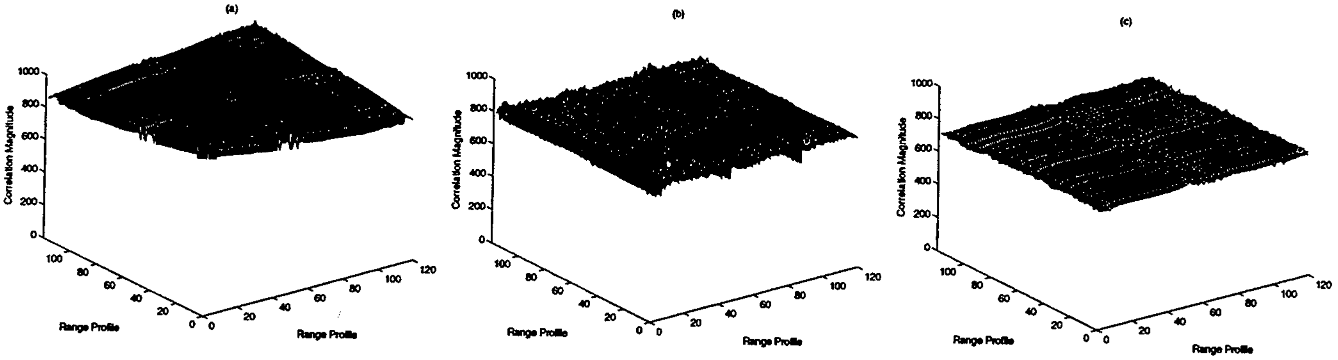


Figure 12: A naive identification test. The absolute inner product of pairs of range profiles from the DC-10 data set is shown in (a). The absolute inner product of each range profile from the DC-10 and each range profile from the DC-9 is shown in (b). The absolute inner product of each range profile from the DC-10 and each range profile from the Cessna 310 is shown in (c). Because the DC-10 range profiles are more highly correlated with themselves than with range profiles for the other aircraft, these data and the absolute inner product metric could be used in identification.

It is now desired to use these data sets to perform a naive identification test. Suppose that some of the range profiles for each aircraft are stored as a library of templates, and one of the remaining range profiles is selected as an observation. Can the aircraft which produced the observation be identified by simply computing the inner product of the observation with each of the templates? In a more realistic identification test, observations would be compared to templates stored for each aircraft at identical orientations. This could be approximately true for these data if, for example, the aircraft were all illuminated while on identical flight paths or while landing at the same runway. In any case, we will ignore this discrepancy for the purposes of our test.

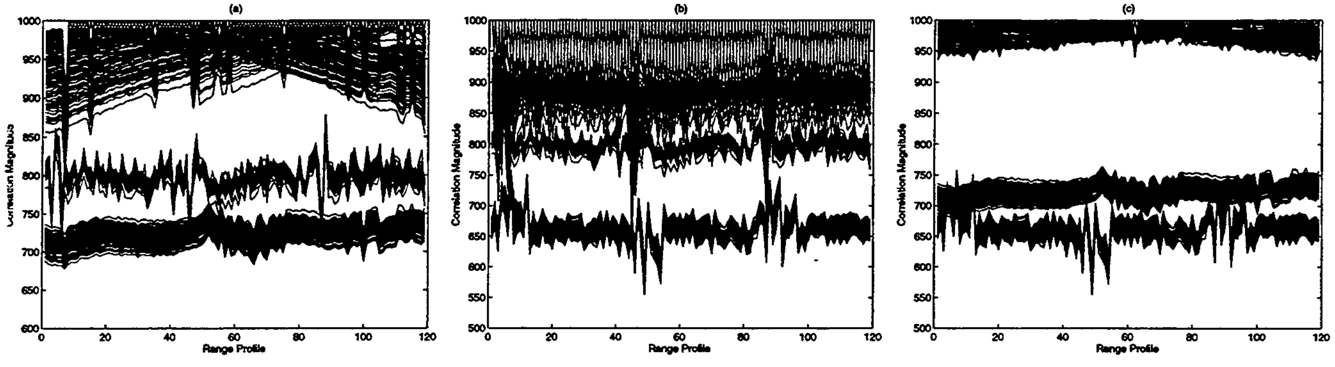


Figure 13: Side view of 3 tests

Before proceeding with the test, note the ridges in the plots of figure 11 which correspond to range profiles of slightly higher energy than their neighbors. Note also that the energy of the range profiles for different aircraft vary by two orders of magnitude. Both of these variations will be eliminated by normalizing each range profile to have an energy of 1000.

Figure 12 shows the results of the first identification test. The absolute inner product between pairs of Doppler-corrected, normalized range profiles from the DC-10 data set is plotted in 12 (a). The absolute inner product of each DC-10 range profile with each range profile from the DC-9 and Cessna 310 data sets is shown in 12 (b) and 12 (c), respectively. Note that the DC-10 range profiles are more highly correlated with each other than they are with those from the other two aircraft. We will therefore say that these data and the absolute inner product metric could be used for identification, subject to the caveats mentioned previously.

The difference between the three plots of figure 12 can be seen by plotting the rows of all three inner product matrices on the same set of axes, as in figure 13 (a). The upper group of traces in this plot correspond to the self-correlation of the DC-10 range profiles, while the middle and lower groups of traces correspond to the cross-correlation of the DC-10 data with the DC-9 and Cessna 310 data, respectively. Note that the upper group of traces is almost completely separated from the lower two groups, indicating that range profiles from the DC-10 data set are consistently more highly correlated with each other than they are with range profiles from the other aircraft. The one exception to this rule is range profile number 7 from the DC-10 data set, which has exceptionally low correlation with others for the same aircraft. Examination of the original data indicates that range profile number 7 may be shifted in range relative to others in the DC-10 data set, resulting in the low correlation.

Figure 13 (b) shows the results of correlating range profiles from the DC-9 data sets with all three data sets. The uppermost group of traces is again the self-correlation data, followed by the cross-correlation with the DC-10 and Cessna 310 data sets, respectively. Note that the separation seen previously is not evident here; Certain DC-9 range profiles are less highly correlated with others than they are with some DC-10 range profiles. Thus, it may not be possible to perform identification on an observed DC-9 range profile by correlating with templates for various aircraft. It should be noted that the DC-9 range profiles did not exhibit the consistency in magnitude which was shown in figure 9 for the Cessna 310 data. Examination of this data set indicates that the range profiles are corrupted by a periodic modulation, the source of which is not known.

Figure 13 (c) shows an analogous identification test for the Cessna 310 data set. Here the Cessna range profiles are much more highly correlated with each other than with those of the DC-10 or DC-9. Such behavior might be expected as the Cessna differs from the other aircraft in size, propulsion, maximum velocity, and a number of other factors, although the precise nature of the discrepancy cannot be inferred from examination of these data sets.

.Examination of figure 12 indicates that the absolute inner product of the first and last range profiles has a value which is approximately 0.85 times the energy in these range profiles. It would be desirable to determine the change in the orientation of the aircraft during the collection of these data, in order to estimate the orientation change which would produce decorrelation in the range profiles. The data files from which these range profiles were extracted each have a header which indicates the position of the aircraft relative to the radar system during each illumination interval. It was hoped that analysis of the header data would allow for estimation of the orientation change. For a variety of reasons, such analysis has proven elusive. We therefore assume without justification the following data collection scenario: the DC-10 is flying directly toward the radar system with a velocity of 500 miles per hour, at a nominal distance of 10 miles and a nominal elevation angle of 30 degrees during illumination, and the complete set of 119 range profiles were collected over an interval of one second. Using these assumed values, trigonometric calculations yield an orientation change during data collection of 0.4 degrees. Thus we would expect that these range profiles would decorrelate over an orientation change on the order of 1 to 2 degrees. While it is difficult to have confidence in these numbers due to the ad hoc assumptions made in producing them, if they are roughly correct they would indicate that the variation of the range profile with changes in orientation is considerably less volatile than that which was predicted for complex targets by analysis of simulated data.

5 Proposal of Future Work

This document has summarized the research I have done in investigating the problem of Automatic Target Recognition and the role of High Range Resolution radar data in performing orientation estimation and identification tasks. A radar reflectivity model was developed which may be used to compute the expected variation of the range profile with target orientation. Simulated range profiles were produced using the software package xpatch and the variation of the data for small changes in orientation was studied. Actual HRR radar range profiles were analyzed to determine the extent to which they are self-consistent over multiple illuminations of the same target, and the extent to which they are target specific. The results have provided very limited insight into some fundamental issues related to the successful implementation of algorithms capable of recognizing aircraft from their radar reflections.

Further work to investigate these questions would begin with extensions to the work already done. Completion of the radar reflectivity model based on random arrays to yield the expected variation of the range profile of a complex target with small changes in orientation is highly desirable. The primary obstacle to completion of this task is the issue of integration over the target surface.

Additional simulation studies using xpatch would investigate the variation in the range profiles in the vicinity of a number of nominal orientations. Implementation of exhaustive and gradient-based searches on these data for orientation estimation are clearly indicated.

Further analysis of real range profiles is also warranted. Knowledge of the aircraft position and orientation during each illumination would be very useful for understanding the variation in real radar data as the orientation of the aircraft changes during illumination. Work is progressing toward development of a library of range profiles simulated by xpatch for one of the aircraft for which we have real data, with the goal of searching this library to estimate the orientation of the real aircraft during illumination. This in turn requires understanding the conditions under which the range profiles simulated by xpatch can be expected to accurately represent the real data.

Interpolation of range profiles is an important issue which has not been addressed in this document. In implementing a gradient-based search for orientation estimation, knowledge of the range profiles corresponding to orientations which are not grid points is required. If the density of grid points is high enough, then range profiles in between these points can be found through an appropriate interpolation algorithm. Of particular interest is the grid density required such that range profiles found by interpolation are a sufficiently accurate representation of those resulting from direct simulation. Possible measures of quality for interpolated range profiles include the squared magnitude of the error between interpolated and simulated range profiles, as compared to the energy in the range profiles. Alternatively, we may investigate the grid density required such that an interpolated range profile is closer to one obtained by direct simulation than are any of the range profiles corresponding to the grid points from which the interpolation was performed. In addition to direct analysis of simulated data, it would be extremely useful if the random array model could be used to make theoretical predictions regarding the grid density required for successful interpolation.

The analysis performed by Ksienski and White was important in that it was the first attempt to examine the radar data surface corresponding to a given target at all orientations, and to investigate the target-specific nature of the data by examining the extent to which data surfaces for different targets occupy mutually exclusive regions in the data space. A similar study of the intersection and proximity of range profile surfaces for multiple targets is desired. The grid density necessary for interpolation would be used in this study also, to ensure that analysis of the intersection and proximity of approximated data surfaces is representative of the properties of the complete surfaces. An important result of this analysis would be the accuracy of orientation estimates required such that identification could be successfully performed.

The ultimate goal of this research is clearly the successful implementation of orientation estimation and identification tasks using real or simulated range profiles, and the incorporation of these tasks into a larger system capable of detecting, tracking and identifying multiple dynamic aircraft over time. Work is progressing on this development. The analyses performed in the preparation of this document and those proposed in this section will aid in understanding how a successful implementation may be achieved and in predicting the performance that will result.

References

[1] Narendra S. Goel, editor. *Remote Sensing Reviews: Automatic Target Recognition and Tracking Based on Shape Analysis*, volume 6. Harwood Academic Publishers, 1992.

[2] Joseph W. Goodman. *Introduction to Fourier Optics.* McGraw-Hill, 1988.

[3] Don H. Johnson and Dan E. Dudgeon. *Array Signal Processing.* Prentice Hall, 1993.

[4] Edmund W. Libby and Peter S. Maybeck. Application of sequence comparison techniques to multisensor data fusion and target recognition. In *Proceedings of the 32nd Conference on Decision and Control*, December 1993.

[5] Edward W. Libby. *Applicstion of Sequence Comparison Methods to Multisensor Data Fusion and Target Recognition.* PhD thesis, Air University, June 1993.

[6] Aharon A. Ksienskiand Yau-Tang Lin and Lee James White. Low-frequency approach to target identification. *Proceedings of the IEEE*, 63(12):1651–1659, December 1975.

[7] Michael I. Miller, Anuj Srivastava, and Ulf Grenander. Conditional mean estimation via jump-diffusion processes in multiple target tracking/recognition. Accepted for publication in IEEE Transactions on Signal Processing.

[8] Michael I. Miller, Robert S. Teichman, Anuj Srivastava, Joseph A. O'Sullivan, and Donald L. Snyder. Jump-diffusion processes for automated tracking-target recognition. In *1993 Conference on Information Sciences and Systems*, Baltimore, Maryland, March 1993. Johns Hopkins University.

[9] Joseph A. O'Sullivan, Steven P. Jacobs, Michael I. Miller, and Donald L. Snyder. A likelihood-based approach to joint target tracking and identification. In Avtar Singh, editor, *Conference Record of the Twenty-Seventh Asilomar Conference on Signals, Systems & Computers*, pages 290–294, Los Alamitos, California, November 1993. IEEE Computer Society Press.

[10] J. A. Shapiro, B. A. Capron, and R. C. Harney. Imaging and target detection with a heterodyne-reception optical radar. *Applied Optics*, 20(19):3292–3313, October 1981.

[11] Jeffery A. Shapiro. Target-reflectivity theory for coherent laser radars. *Applied Optics*, 21(18):3398–3407, September 1982.

[12] Jeffery H. Shapiro. Correlation scales of laser speckle in hetrodyne detection. *Applied Optics*, 24(12):1883–1888, June 1985.

[13] Anuj Srivastava, Nicholas J. Cutaia, Michael I. Miller, Joseph A. O'Sullivan, and Donald L. Snyder. Multi-target narrowband direction finding and tracking using motion dynamics. In *Proceedings of the 30th Annual Allerton Conference on Communication Control and Computing*, pages 279–288, Urbana, Illinois, October 1992. University of Illinois.

[14] Bernard D. Steinberg. *Principles of Aperture and Array System Design*, chapter 8. John Wiley and Sons, 1976.

[15] L. J. White and A. A. Ksienski. Aircraft identification using a bilinear surface representation of radar data. *Pattern Recognition*, 6:35–45, 1974.

V. Software

In this section, we provide a complete listing of the code developed for Rome Laboratory as part of this project. The code is fully commented. The programs run in a distributed computing environment in our laboratory. Most of the programs run on Silicon Graphics machines. Others run on our Maspar 12000 (the 128 by 128 array of processors that are used by the recognition and tracking algorithms for the computationally intensive parts of the algorithms). The communications between the sections of code are documented along with the data flow.

JOINT TARGET TRACKING AND IDENTIFICATION DEMONSTRATION

INTRODUCTORY INFORMATION

The software package documented here is a demonstration of a system for joint target tracking and identification using multiple radar sensors. Ultimately, our envisioned system will perform detection, tracking, orientation estimation, and target type identification in a multiple target scenario. The system documented here performs detection, tracking, and orientation estimation for a single target.

The general structure of this software system is displayed in three figures included with the documentation. As indicated in figure 1, this system actually consists of three programs which run simulataneously. Data simulation and estimation of all target parameters is perfromed by a program on the DEC MPP. Two separate programs that display the results of tracking and orientation estimation, respectively, are run on a Silicon Graphics workstation. Sockets are established for communication of true and estimated parameters from the MPP estimation algorithm to the SGI visualization programs.

Figure 2 provides a simplified flow diagram of the estimation algorithm. After initializing communication sockets, the true track for the target is read from a disk file. This file is produced using a Silicon Graphics flight simulator, and consists of a history of positions and orientations for the target at discrete time instants.

For each segment in the track, data from a cross array of radar sensors is simulated in real-time using the true position of the target. Tracking of the target is performed using jump-diffusion processes. A jump move is randomly selected by the algorithm. In this demonstration, the jump move may consist of a track birth, which corresponds to detection of a new target, segment birth, in which an existing track is extended by one segment, or segment death, in which an existing track is reduced by one segment. This jump move is accepted or rejected by evaluating the data loglikelihood with and without the proposed jump move. Diffusions are performed on the last 64 estimated target positions to maximize the loglikelihood of the tracking data.

An observed range profile is simulated by selecting from a library of range profiles which have been precomputed for a discrete set of orientations of the target. The range profile corresponding to the discrete orientation nearest the true orientation is selected as the observed range profile. Orientation estimation is achieved by performing a gradient-based search through the library in order to minimize the squared error, summed over all range bins, between observed and library-selected range profiles.

Figure 3 provides a more detailed desciption of the jump-diffusion precess as it applies to the general scenario of joint target detection, tracking and identification. Note that in this figure, the additional jump moves of track death, which corresponds to deleting a hypothesized target from the scene, and change of target type, which is only applicable in systems which perform target identification, are included.

Further explanation of the operation of this software package is

Figure 1

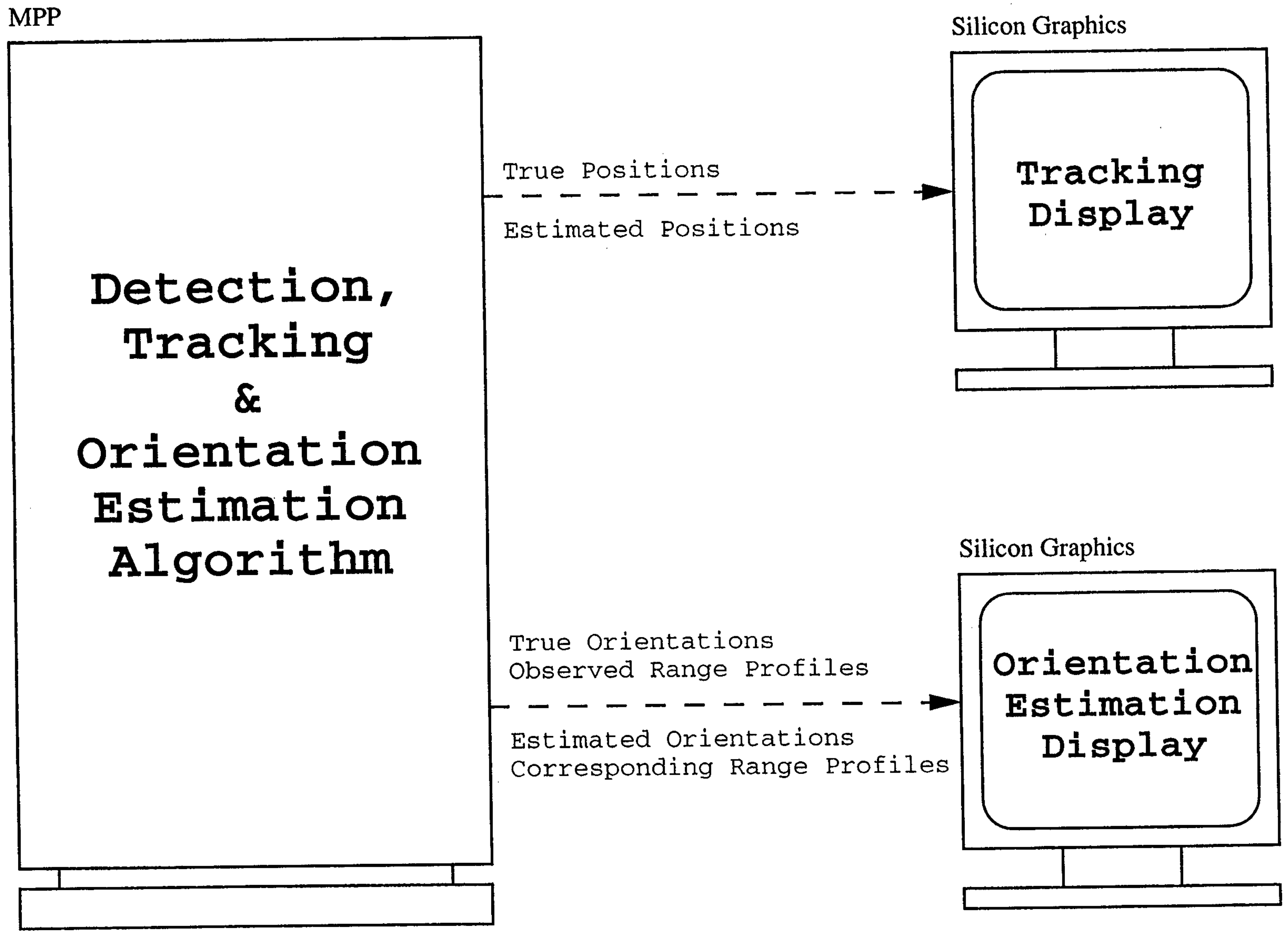
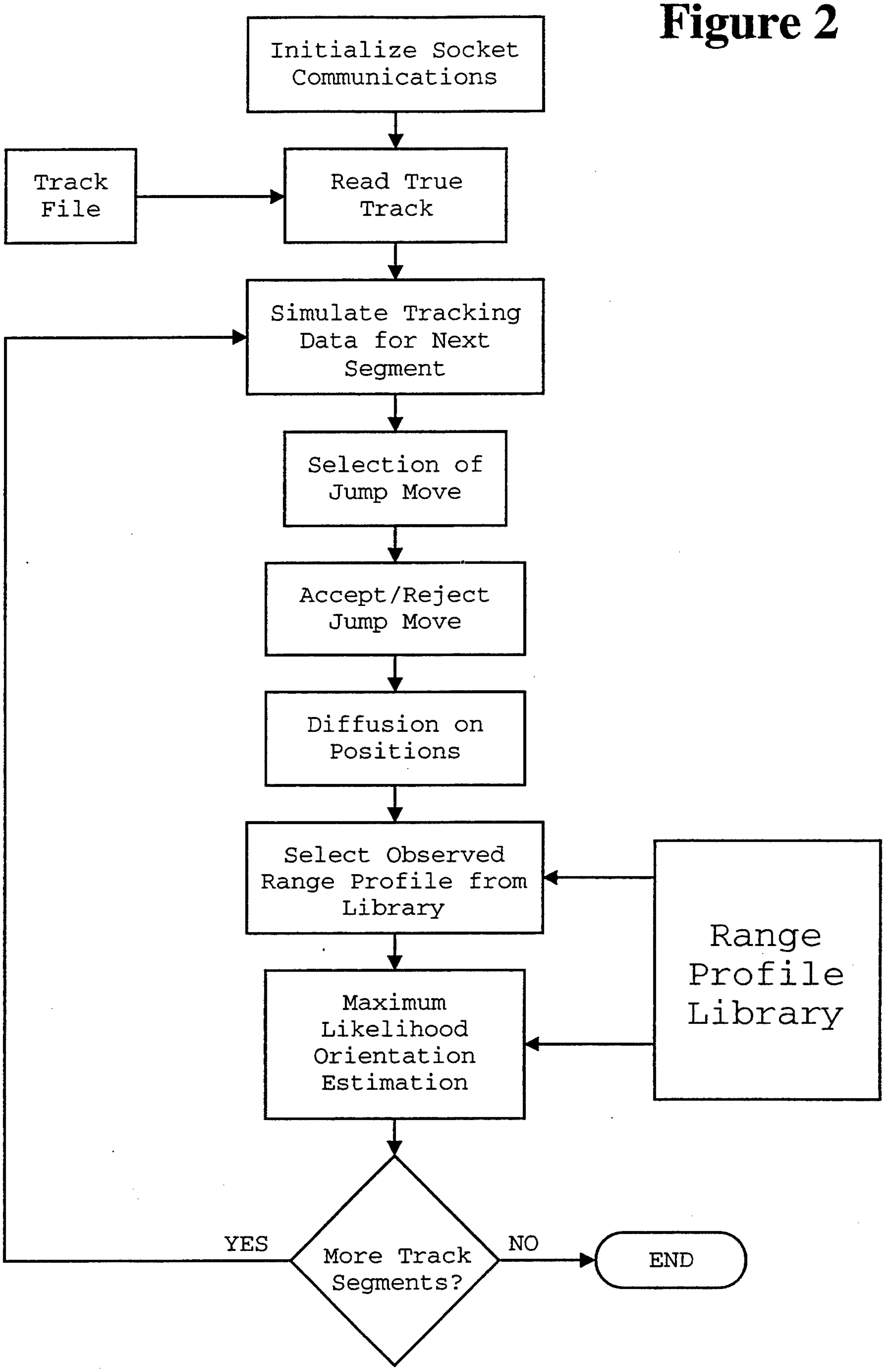
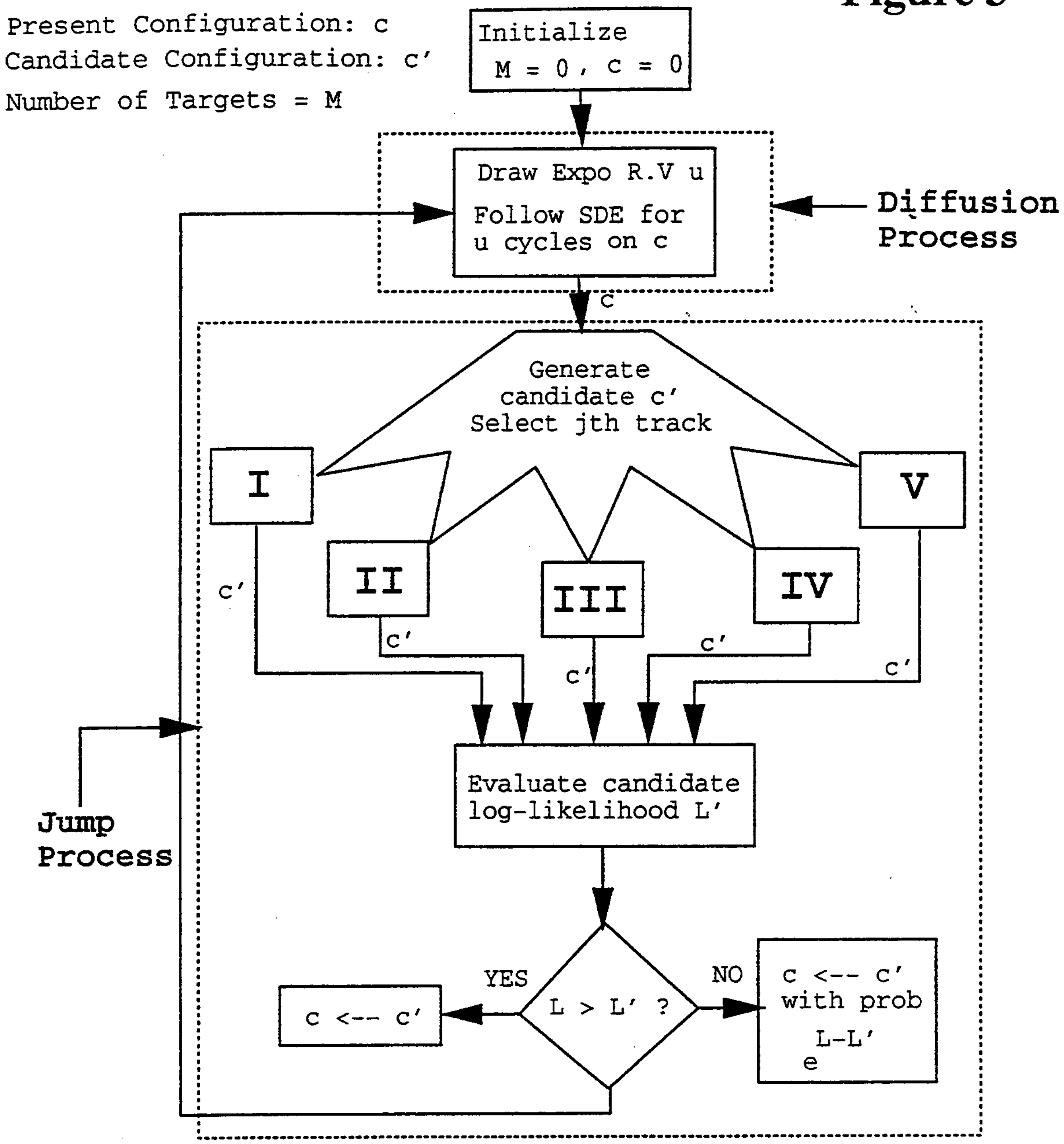

MPP
Detection, Tracking & Orientation Estimation Algorithm
True Positions
Estimated Positions
Silicon Graphics
Tracking Display
True Orientations
Observed Range Profiles
Estimated Orientations
Corresponding Range Profiles
Silicon Graphics
Orientation Estimation Display

Figure 2



Initialize Socket Communications
Track File
Read True Track
Simulate Tracking Data for Next Segment
Selection of Jump Move
Accept/Reject Jump Move
Diffusion on Positions
Select Observed Range Profile from Library
Range Profile Library
Maximum Likelihood Orientation Estimation
YES
More Track Segments?
NO
END

Figure 3



I. Track Death: Remove jth track from c.

II. Track Birth: Add a new track to c.

III. Segment Death: Remove last segment from jth track.

IV. Segment Birth: Add one segment to jth track.

V. Change Target Type: Change target ID of jth track.

The current directory is

~atr/demo

This is the top directory containing a software package which performs detection, tracking and orientation estimation on a single target using simulated data. The only source code in this directory is the main mpp program called main.m. This program performs some initializations and calls other routines to simulate data and estimate target parameters. These routines are contained in the following subdirectories:

anujsrc/	Contains mpp tracking routines.
display/	Contains C program big.c which displays tracking results.
faisalsrc/	Contains mpp routines for orientation estimation.
include/	Contains all header files.
marksrc/lib/	Contains mpp target detection routines.
sgi/	Contains C program display_rps.c which displays results of orientation estimation.

Further information is available in the subdirectories.

```
/************************************************************************
   main.m
*************************************************************************
   These comments last edited 6/13/95 by Jason August (jta1@cec.wustl.edu)
   main.m is the first part of the tracking/recognition program.
   It does the following:
      1) read in the track-data file from the SGE flight simulator.  This
         data is only used to show actual vs. predicted paths when displayed
         for testing purposes, and is not seen by the tracking and
         recognition algorithms.
      2) call initialization.m to set up network communication sockets
         and to read the object templates file which contains all
         information known about objects to be tracked (except their
         path and location of course)
** End Introductory Comments */

/************************************************************************
   Header Files to be included...
************************************************************************/
#include <stdio.h>        /* standard i/o header */
#include <math.h>         /* math header */
#include <mpl.h>          /* MasPar programming language header */
#include <mpml.h>         /* MasPar mathematics library header */
#include <ppeio.h>        /* parallel processing environment i/o header */
#include <fcntl.h>        /* file system control header */
#include <su/values.h>    /* DEC header file of common values */
#include <su/stdlib.h>    /* DEC std. library.h (/usr/maspar/include/su) */
#include <head.h>         /* the big header file that everything is in */
#include <faisalheader.h>/*header for orientation estimation routines */

/* End Header Files to be included */

/************************************************************************
 Global Variable Declarations
************************************************************************/

plural float index_u,          /* an index matrix                     */
             index_d,          /* an index martix                     */
             JumpRand;         /* random jump matrix                  */

int    send_to_jupiter_1995,      /* socket 1995 sends display data      */
       send_to_jupiter_1991,      /* socket 1991 also sends display data */
       recv_from_jupiter_1992,    /* socket 1992 gets display feedback   */

       num_obj = 0,            /* number of objects detected when     */
                               /* simulation begins: 0                */
       do_jupiter = 1;         /* display the data                    */

float track[OBJNUM_MAX][tracksize][3], /* position history of target */
      pitch[OBJNUM_MAX][tracksize],    /* orientation history        */
      yaw[OBJNUM_MAX][tracksize],
      roll[OBJNUM_MAX][tracksize];

float next_orient_pl[3], next_orient_vw[3];

/****** Global Variables for Orientation Estimation ********************/

int num_yaw, count, bins,
    total_num_rp, rp_per_layer, layers,
        p_count, y_count, mem, proc_num,
     are_rps_a_file, want_to_save_dict,
     rp_count, fd;

 int *pitch_vect, *sum_pitch_vect, k;

 char dir_name[200], data_filename[200],
      rp_dict_filename[200], dummy_string[200];

 float init_pitch, init_yaw, delta_yaw,
       f_pitch, f_yaw, dummy, val,
       found_pitch_prime, found_yaw_prime,
       init_roll, init_pitch_prime, init_yaw_prime,
       found_pitch, found_yaw, found_roll;

 float *delta_pitch_vect, typ, tpp,
       flag, z1, z2;

 float true_pitch, true_yaw, true_roll,
       true_pitch_prime, true_yaw_prime;

 int true_proc_num, true_mem;

 unsigned char test_flag;

 plural float *rp_dictionary, *f_data,
              noise_re, noise_im;

 int f_sock;

/******* End Global Variable Declarations *********************************/

visible extern int sendinit_jupiter();

/****************************************************************************
   main:
   This function starts all of the program's processing that runs on the
      MasPar and it's Alpha front end (currently theseus)
***************************************************************************/

main(int argc, char *argv[]){

        int i, r;                    /* loop variables                        */
        char name[80];               /* true track file name                  */
        FILE *fp;                    /* true track file pointer               */
        float scale;                 /* scaling factor for the random value   */
                                     /* that temporarily times jump moves     */
        plural float uniform_rv,     /* random matrices used for jump moves   */
                     expo_rv;

/* if there are > 1  command line parameters, don't send data to tracking */
/* display program */

        if (argc > 1) do_jupiter = 0;

        printf("Reading from the binary file \n");

/* Read real track from binary file; it will be sent to the display */

        for(r=1;r< OBJNUM_TRU+1; r++)
        {
```

```
        sprintf(name,"../Flight_Sim/Cessna.%d.bin",r+1);
                fp = fopen(name,"r");

                i = fread(track[r],sizeof(float),3*(tracksize),fp);
                i = fread(pitch[r],sizeof(float),(tracksize),fp);
                i = fread(roll[r],sizeof(float),(tracksize),fp);
                i = fread(yaw[r],sizeof(float),(tracksize),fp);

                printf("Read file value %d\n",r);

                for (i=0;i<tracksize;i++)
                        track[r][i][0] += 9000.0;

                fclose(fp);
        }

/****** Program Initializations ******************************************/

        track_init();
        init_jupiter();
        f_init_jupiter();

/****** Exponential RV for jump-moves ************************************/

    scale = 2147483647.0;
    uniform_rv = p_random()/scale;
    expo_rv = -fp_log(uniform_rv)*mean_diff;

/* Estimation algorithm is contained in function "base" */

        base(expo_rv,uniform_rv);

/* Free allocated memory and close socket */

        p_free(f_data);
        p_free(rp_dictionary);

        close(f_sock);

}
```

```
The current directory is

        ~atr/demo/anujsrc

This directory contains routines for performing target tracking
using simulated data from a cross-array of radar sensors.
The routines are described individually:

1. main.m:      Performs initialization, reads in the track-data file
                (from SGI flight simulator), calls initialization.m for
                setting up the socket communications and reading the
                templates file.  After initialization base.m takes over.

2. base.m:      This is the file which controls the flow of the algorithm.
                It calls other subroutines like jumps, diffusions, ....
                The important tasks being done in base.m are:

                a. Sending track-input over to SGI for display.

                Inside the iterative loop:

                b. Generates the tracking data (data.m) and receives
                   the imaging data from SGI.

                c. Selecting the type of jump move, executing it.
                   It uses routines like_calculation.m, jump_process.m

                d. Calls diff.m for num_diff (number of diffusion) cycles
                   of the diffusion process for the current track.

                e. Calculates and maintains the prior covariance matrices
                   from equations of motion for prior on tracks. This uses
                   files covariance.m and reshaping_covariance.m.

3. data.m :     Generates the tracking data sets

4. diff.m :     Performs diffusion cycles. It calculates the gradients
                of the likelihood energy and all the chain rule
                combinations required.

5. jump_process.m:
                Calculates the candidate for the next track segment given
                that a segment birth move has been chosen. Uses the equations
                of motion to perfrom candidate selection.

6. convert.m:   Performs various conversions required to shift between
                various parameters:
                positions -> velocities
                velocities -> positions
                rectangular -> spherical coordinates

7. write_socket.m:
                Sends results back to SGI for display at every iteration.

8. GaussRand.m: Generates Gaussian pseudo-random nubers.

9. covariance.m:
                Computes covariance of posisition vector.

10. init_jupiter.m:
```

Initializes local socket connection for tracking results, sends true track over socket.

11. initialization.m:
    Initializes long distance socket connection for tracking results, sends true track over socket.

12. like_calculation.m:
    Computes the likelihood for various potential jump moves.

12. matrix_mult.m:
    Parallel implementation of linear algebra operations.

13. reshaping_covariance.m:
    Reorders entries in covariance matrices.

14. shiftwindow.m:
    Shifts the 64-length track window.

See the source code for further comments.

```
/***************************************************************************
   GaussRand.m
****************************************************************************
   This program assigns random values to the array gauss1 according to
   Gaussian distribution.  The mean of the value is 0 and the variance is
   1.  It transforms 2 uniform random numbers to a Gaussian random number
   for each element.*/

/***************************************************************************
   Header Files to be included...
***************************************************************************/
#include <stdio.h>       /* standard i/o header */
#include <mpl.h>         /* MasPar programming Language header */
#include <math.h>        /* math header */
#include <su/values.h>   /* DEC header file of common values */
#include <su/stdlib.h>   /* DEC std. library.h (/usr/maspar/include/su) */

/***************************************************************************
   Function: GaussRand
***************************************************************************/
void
GaussRand(plural float *gauss1)
{
   plural float rand,  /* the random number, between 0 and 1 */
                theta, /* a uniform random angle */
                r;     /* Rayleigh distributed random radius */
   float        pi,
                scale; /* scales p_random to between 0 and 1 */

   pi = 4.0*atan(1.0);
   scale=2147483647.0;

   theta=2*pi*(p_random()/scale);

   rand=p_random()/scale;
   while (rand==0){
      rand=p_random()/scale;
   }
   r=fp_sqrt(-2.0*fp_log(rand));

   *gauss1=r*fp_cos(theta);
}
/* End Function: GaussRand */
```

```
/************************************************************************
  base.m
*************************************************************************
  This is the main program that is executed on the MasPar.  It calls the
   other programs...
********************************************** End Introductory Comments */

/************************************************************************
  Header Files to be included...
************************************************************************/
#include <stdio.h>       /* standard i/o header */
#include <math.h>        /* math header */
#include <mpl.h>         /* MasPar programming language header */
#include <mpml.h>        /* MasPar mathematics library header */
#include <ppeio.h>       /* parallel processing environment i/o header */
#include <fcntl.h>       /* file system control header */
#include <stdlib.h>      /* standard library */
#include <values.h>      /* DEC header file of common values */
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/file.h>
#include <su/unistd.h>

#include <head.h>        /* the big headerfile that everything is in */
#include <param.h>       /* system parameter header file */
#include "../marksrc/lib/head.h"/* more of the stuff like head.h */
#include "../include/main.h"
#include "../include/faisalheader.h"

/* End Header Files to be included */

/************************************************************************
  Global Variable Declarations
************************************************************************/

extern          plural float index_u,index_d;
extern          int send_to_jupiter_1995, send_to_jupiter_1991;
extern          int recv_from_jupiter_1992;
extern          float track[OBJNUM_MAX][tracksize][3],
                        pitch[OBJNUM_MAX][tracksize],
                        yaw[OBJNUM_MAX][tracksize],
                        roll[OBJNUM_MAX][tracksize];
extern          plural int rob_data[OBJNUM_MAX/2];
extern          int num_obj, do_jupiter;
extern          float true_pitch,true_yaw,true_roll;
extern          float init_pitch, init_yaw, init_roll;
extern          float found_yaw, found_pitch, found_roll;

/************************************************************************
   function: track_init
   form index matricies for the tracks
************************************************************************/
void track_init()
{
/* Forming index matrices       */
                index_u = index_d = iyproc ;
                index_u = index_u - 15.5;
                index_d = index_d - 47.5;
                        if(iyproc > 31)
                            index_u = 0.0;
```

```
                          if(iyproc < 32)
                               index_d = 0.0;
}

/***************************************************************************
   function: base
      initialize variables
      initial data generation
      select track
      choose jump move -
         segment birth, segment death,
         track birth (object detection)
      do jump process (see if choice is acceptable)
      diffusion
***************************************************************************/
void base(plural float expo, plural float uniform_rv)
{
        int j,n,r,len_track,i,k,num_diff,shift_yes,flag=1,option=0,DELETE_TRY;
        int n_obj[OBJNUM_MAX],sel,read_indx,max_indx,jump_mode,jump_mode_bak;
        plural int len_mask;
        float pi,X0[OBJNUM_MAX],Y0[OBJNUM_MAX],Z0[OBJNUM_MAX], Prob_jump;
        int len_obj[OBJNUM_MAX],obj_num_est, move_type, accept_reject;
        plural float range_dat;
        plural float theta,phi,psi;
        plural float T[3][3];
        plural float cov_in[OBJNUM_MAX][3][2];
        plural float cov_out[OBJNUM_MAX][3][2];
        plural float K_x[OBJNUM_MAX][3][2];
        plural float dat_r,dat_i;
        plural float u,v,w;
        plural float X,Y,Z;
        plural float alpha[OBJNUM_MAX/2],beta[OBJNUM_MAX/2];
        double timer_sample(),timed_data;
        float p_rate[OBJNUM_MAX],q_rate[OBJNUM_MAX],r_rate[OBJNUM_MAX];
        float flip,likeli,like_d,like_n,like_b;
        float rot_mat[3][3];
        plural int datum[OBJNUM_MAX];

        float jump_rv;
        int jump_choice;

/***** TRACK_BIRTH variables ******/

        plural float tmp_Data[2], Data[2], candidate;
        float cand_el,cand_az,X_temp,Y_temp,Z_temp,Cand_Range,pos[OBJNUM_MAX][3];
        int index[OBJNUM_MAX];
        float Xinit[OBJNUM_MAX],Yinit[OBJNUM_MAX],Zinit[OBJNUM_MAX];
        plural int rej_mask = 1;

        /* initialize variables */
        pi = 4.0*atan(1.0);
        theta = phi = psi = 0.0;
        dat_r = dat_i = 0.0;
        u = v = w = 0.0;
        len_track = 1500;
        read_indx = -1;
        len_mask = 0;
        move_type = accept_reject = 0;

        sel = 1;
        for(r=0;r<OBJNUM_MAX;r++)
        {
                n_obj[r] = -1;
                len_obj[r] = -1;
                index[r] = 0;
        }

        for(r=0;r<num_obj; r++)
        {
                X0[r] = track[r][0][0];
                Y0[r] = track[r][0][1];
                Z0[r] = track[r][0][2];
        }

        printf("BASE: done initializing variables \n");

/* Initial data generation for track birth */

        max_indx = -1;

        printf("BASE: ready to call DATA\n");
    data(len_obj,max_indx,&dat_r,&dat_i);
        printf("BASE: returning from DATA\n");

    Data[0] = 0;
        Data[1] = 0;

    Data[0] = xnetE[64+max_indx].dat_r;
    Data[1] = xnetE[64+max_indx].dat_i;

        for(j=0;j<STOPF+1; j++)
        {
                printf("Simulation Iteration = %d ********\n",j);
                shift_yes = 0;

/*** Selecting the track ****/

                for(r=0;r<num_obj;r++)
                        read_indx = max(read_indx,len_obj[r]);

                printf("\t Est: ");
                for(r=0; r <num_obj; r++)
                        printf(" %d-%d",r,len_obj[r]);
                printf("\n");


                if (do_jupiter) write_socket(num_obj,send_to_jupiter_1995, len_obj, n_obj,

                                len_track,move_type, accept_reject,  X, Y, Z, index);

                /*** choice of a jump move (random for now; each has the same
             probablility) ***/

                srandom(get_rand_seed());

                /* select jump (track) value for speed improvement */

                if(num_obj < OBJNUM_TRU)
```

```
                        jump_rv = 0.8;
                else
                        jump_rv = (random()*1.0)/MAXINT;

                /* start selecting... */
                if ((jump_rv < 1.0) && (num_obj > 0))
                {
                        jump_choice = SEG_BIRTH;
                        move_type = 1;
                        printf("\t JUMP MOVE: SEG_BIRTH\n");

                else if ((jump_rv < 0.8) && (num_obj > 0))
                {
                        jump_choice = SEG_DEATH;
                        move_type = 2;
                        printf("\t JUMP MOVE: SEG_DEATH\n");
                }
                else if ((jump_rv < 1.0) || (num_obj == 0) && num_obj < 3)
                {
                        jump_choice = TRACK_BIRTH;
                        move_type = 3;
                        printf("\t JUMP MOVE: TRACK_BIRTH\n");
                }

        if(jump_choice == SEG_DEATH)
                {
                        if(len_obj[sel] > 3)
                        {
                                DELETE_TRY = 1;
                                printf("\t Trying to Delete segment %d of track %d\n",
                                        len_obj[sel]-1,sel);

                                /*** Deletion of segment ***/
                                option = 1;
                        like(option,sel,n_obj,dat_r,dat_i, range_dat,alpha,beta,
                                                                &like_d, X,Y,Z);

                                /*** No addition or deletion *****/
                                option = 2;
                        like(option,sel,n_obj,dat_r,dat_i, range_dat,alpha,beta,
                                                                &like_n, X,Y,Z);

                if(-like_d > -like_n)
                                {
                                        /*** No segment added or deleted in this option **
***/
                                        printf("\t \t ---Didn't accept the Delete option\n
");

                                        /*** Compensate for the n_obj values for other tra
cks ****/
                                        accept_reject = 2;
                                        }
                        else
                                {
                                        accept_reject = 1;

                                        /*** Segment deleted from end of sel^th track ****
/
```

```
                printf("\t \t  Deleting %dth segment from track %d\n",
                        len_obj[sel]-1, sel);
                len_obj[sel] = len_obj[sel]-1;
                n_obj[sel] = n_obj[sel]-1;
                }
        }
        else
                accept_reject = 2;
}
else if (jump_choice == SEG_BIRTH)
{
        /*** Selecting the track ****/
        if (num_obj > 0)
                sel = (sel + 1) % num_obj;

        DELETE_TRY = 0;

        printf("\t Track index for segment birth = %d\n",sel);
        n_obj[sel]++;
        len_obj[sel]++;

if(n_obj[sel] > 63)
        {
        shift_yes = 1;
        n_obj[sel] = 63;

        dat_r = xnetE[1].dat_r;
        dat_i = xnetE[1].dat_i;

        u = xnetS[1].u;
        v = xnetS[1].v;
        w = xnetS[1].w;

        for(r=0;r< num_obj; r++)
                {
                X0[r]  = proc[0][r].X;
                Y0[r]  = proc[0][r].Y;
                Z0[r]  = proc[0][r].Z;
                }

        theta = xnetS[1].theta;
        phi   = xnetS[1].phi;
        psi   = xnetS[1].psi;
        range_dat = xnetS[1].range_dat;

        if(ixproc == 63)
                dat_r = dat_i = 0.0;

        if(iyproc == 63)
                {
                u = v = w = 0.0;
                X = Y = Z = 0.0;
                theta = phi = psi = 0.0;
                range_dat = 0.0;
                }

        for(r=0;r<num_obj;r++)
                n_obj[r] = n_obj[r] - 1;

        n_obj[sel] = n_obj[sel]+1;
```

```
                }

/**** Complete Data Generation ******/

                        max_indx = -1;
                        for(r=0;r<num_obj;r++)
                                max_indx = max(max_indx,n_obj[r]);

                        data(len_obj,max_indx,&dat_r,&dat_i);
                        range_data(index, len_obj, max_indx, &range_dat);

                for(r=0;r<num_obj;r++)
                        {
                        if(iyproc <= n_obj[r] && ixproc == r)
                                len_mask = 1;
                        else if(iyproc > n_obj[r] && ixproc == r)
                                len_mask = 0;
                        }

/************** Setting the rotation angles provided *****/

                if(iyproc == n_obj[sel] && ixproc == sel)
                        {
                        if(pitch[sel][len_obj[sel]] < 0)
                                theta = pitch[sel][len_obj[sel]]*pi/180.0 + 2*pi;
                        else if(pitch[sel][len_obj[sel]] > 360.0)
                                theta =pitch[sel][len_obj[sel]]*pi/180.0 - 2*pi;
                        else
                                theta = pitch[sel][len_obj[sel]]*pi/180.0;

                        if(yaw[sel][len_obj[sel]] < 0)
                                psi = yaw[sel][len_obj[sel]]*pi/180.0 + 2*pi;
                        else if(yaw[sel][len_obj[sel]] > 360.0)
                                psi =yaw[sel][len_obj[sel]]*pi/180.0 - 2*pi;
                        else
                                psi = yaw[sel][len_obj[sel]]*pi/180.0;

                        if(roll[sel][len_obj[sel]] < 0)
                                phi = roll[sel][len_obj[sel]]*pi/180.0 + 2*pi;
                        else if(roll[sel][len_obj[sel]] > 360.0)
                                phi =roll[sel][len_obj[sel]]*pi/180.0 - 2*pi;
                        else
                                phi = roll[sel][len_obj[sel]]*pi/180.0;
                }

                form_sin_cos(len_mask,theta,phi,psi,T);


/****** Jump process: to find nth position or n-1th velocity ******/

                /*
                * appends a segment birth candidate in three body frame velocities
                * with the track index sel
                */
                        jump_metro(n_obj,sel,read_indx,theta,phi,psi,
                                &u,&v,&w,T,p_rate,q_rate,r_rate);

                /*
                * convert velocity histories to inertial frame position histories
                * for all tracks
                */
                        conv(n_obj,len_mask,X0,Y0,Z0,T,u,v,w,&X,&Y,&Z);

                        /*
                * convert position histories from rectangular to spherical
                * coordinates for all tracks
                */
                        short_conv(n_obj,X,Y,Z,alpha,beta,len_mask);

/*** METROPOLIS SELECTION ********************************/
/*** (form cone of possibilities, select segment path) ***/

                        /*** No addition or deletion *****/
                        option = 2;
                like(option,sel,n_obj,dat_r,dat_i, range_dat,alpha,beta,
                                                        &like_n, X,Y,Z);

                        /*** Adding segment to selected track *****/
                        option = 3;
                like(option,sel,n_obj,dat_r,dat_i, range_dat,alpha,beta,
                                                        &like_b, X,Y,Z);

                if(-like_n >= -like_b)
                {
                                /** Candidate has lower energy ***/
                                accept_reject = 1;
                                printf("\t \t--Segment accpeted for birth\n");}
                else
                        {
                        Prob_jump = exp(like_b - like_n);
                                if(random()/2147483647.0 >  Prob_jump)
                                {
                                                DELETE_TRY = 1;
                                        len_obj[sel] = len_obj[sel] -1;
                                        n_obj[sel] = n_obj[sel]-1;
                                                printf("    Candidate Rejected");
                                }
                                accept_reject = 2;
                                printf("\n");
                        }

                        /****** Taking care of the covariances of each track *****/
                        if(shift_yes)
                        {
                                for(r=0;r< num_obj; r++)
                                {
                                        shift_cov_in(r,cov_in);
                                        shift_cov_out(r,cov_in);
                                }
                        }

                        if(n_obj[sel] > -1)
                        {
                                form_cov(n_obj[sel],cov_in,sel,p_rate,q_rate,r_rate);
                                reshape_cov(n_obj,sel,cov_in,cov_out);
                                mymatmul(n_obj[sel],sel,T,cov_out,K_x);
                        }

                        /***** Lower triangle matrix for speed *****/
                for(i=0;i<3;i++)
                        {
                        for(k=0;k<2;k++)
                                {
                                if((iyproc<ixproc && ixproc<64) || (iyproc<ixproc-64 && ix
proc>63))
                                K_x[sel][i][k] = 0.0;
                                }
                        }
                }

                else if (jump_choice == TRACK_BIRTH)
                {
                        for (r=0;r<num_obj;r++)
                        {
                                pos[r][0] = Xinit[r];
                                pos[r][1] = Yinit[r];
                                pos[r][2] = Zinit[r];
                        }

                        max_indx = -1;
                for(r=0;r<num_obj;r++)
                        max_indx = max(max_indx,n_obj[r]);

                        tmp_Data[0] = Data[0];
                        tmp_Data[1] = Data[1];

                        candidate = t_birth(tmp_Data, pos, num_obj, rej_mask);

                        cand_el = proc[0].candidate;
                        cand_az = proc[1].candidate;
                        Cand_Range = range_detect(cand_el, cand_az, index, num_obj);

                        if (Cand_Range > 0)
                        {
                                cand_el = cand_el * pi / 180;
                                cand_az = cand_az * pi / 180;

                                X0[num_obj] = Cand_Range*cos(cand_el)*cos(cand_az);
                                Y0[num_obj] = Cand_Range*cos(cand_el)*sin(cand_az);
                                Z0[num_obj] = Cand_Range*sin(cand_el);

                        Xinit[num_obj] = X0[num_obj];
                        Yinit[num_obj] = Y0[num_obj];
                        Zinit[num_obj] = Z0[num_obj];

                                /*** No addition or deletion *****/
                        option = 2;
                like_track_birth(num_obj,Data[0],Data[1],range_dat,
                                                        &like_n, Xinit,Yinit,Zinit);

                                /*** Adding track *****/
                option = 4;
                like_track_birth(num_obj+1,Data[0],Data[1], range_dat,
                                                &like_b, Xinit,Yinit,Zinit);

                                if(-like_n >= -like_b)
                        {
                                /** Candidate has lower energy ***/
                                printf("\t \t--Object accepted for birth\n");
```

```
                                        num_obj++;
                                        accept_reject = 1;
                                        }
                                else
                                        {
                                        printf("\t Object rejected for birth\n");
                                                proc[(int) proc[1][0].candidate]
                                                                        [(int) proc[1][1].candidat
e].rej_mask = 0;
                                        accept_reject = 2;
                                        }

                                }
                                else
                                {
                                        printf("\t Object rejected for birth\n");
                                        proc[(int) proc[1][0].candidate][(int) proc[1][1].candidat
e].rej_mask = 0;
                                }
                        }

                        /**** Re-evaluating len_mask and n_obj ******/
                        max_indx = -1;
                        for(r=0;r<num_obj;r++)
                                max_indx = max(max_indx,n_obj[r]);

                        for(r=0;r<num_obj;r++)
                        {
                                if(iyproc <= n_obj[r] && ixproc == r)
                                                len_mask = 1;
                                else if(iyproc > n_obj[r] && ixproc == r)
                                                len_mask = 0;
                        }


/********** Diffusion process , gradients over all tracks *****/

                        obj_num_est = 0;
                        for(r=0;r<num_obj;r++)
                                obj_num_est = obj_num_est + proc[max_indx][r].len_mask;

                        if(sel == num_obj-1)
                                num_diff = (proc[j].expo + 20) ;
                        else
                                num_diff = 0;

/*** Orientation estimation using xpatch simulated range profiles ***/

                        true_pitch = pitch[sel][len_obj[sel]];
                        if(true_pitch >= 180.0)
                                true_pitch = true_pitch - 360.0;

                        true_roll = roll[sel][len_obj[sel]];
                        if(true_roll > 180.0)
                                true_roll = true_roll - 360.0;

                        true_yaw = yaw[sel][len_obj[sel]] + 90.0;
                        if(true_yaw > 360.0)
                                true_yaw = true_yaw - 360.0;
```

```
                if(len_obj[sel] == ORSTART)
                {
                        init_pitch = true_pitch;
                        init_yaw = true_yaw;
                        init_roll = true_roll;
                }
                else
                {
                        init_pitch = found_pitch;
                        init_yaw = found_yaw;
                        init_roll = true_roll;
                }

                if(max_indx > -1)
                {
                diff(num_diff,sel,n_obj,len_obj,dat_r,dat_i,range_dat,&X,&Y,&Z,X0,
                        Y0,Z0,K_x,cov_out,len_mask);
                if(len_obj[sel] >= ORSTART && len_obj[sel] < OREND)
                        {
                                if(len_obj[sel]%ORSKIP == 0)
                                {
                                        simulate(len_obj[sel]);
                                        search();
                                }
                        }
                }

        back_conv(len_mask,X,Y,Z,X0,Y0,Z0,T,&u,&v,&w);
        }

}

/************************************************************************
   function: read_sock
   this function reads data from the socket
************************************************************************/
int read_sock(int insock,char *ptr,int sz)
{
        int rd=0;
        int ret;

        while (rd < sz) {
                ret = read(insock,ptr,sz-rd);
                if (ret <= 0) return(ret);
                ptr += ret;
                rd += ret;
        }
        return(rd);
}
```

```
/**************************************************************************
   convert.m
***************************************************************************
   Performs various conversions required to shift between various
   parameters: positions, velocities, azimuth-elevation-range,...
   Forward conversion, backward conversion,..
** End Introductory Comments */

/**************************************************************************
   Header Files to be included...
**************************************************************************/
#include <stdio.h>  /* standard i/o header */
#include <math.h>   /* math header */
#include <mpl.h>    /* MasPar programming language header */
#include <mpml.h>   /* MasPar mathematics library header */
#include <head.h>   /* the big header file that everything is in */
#include <param.h>  /* system parameter header file */
/* End Header Files to be included */

/**************************************************************************
 Global Variable Declarations
**************************************************************************/
extern int num_obj;
/* End Global Variable Declarations */

/**************************************************************************
   function: conv
   This is the main conversion function.  It converts the body frame
   velocity histories for all tracks to inertial frame position histories,
**************************************************************************/
void
conv(int n_obj[OBJNUM_MAX],/* length of each track (0 to 63) */
     plural int len_mask,  /* =1 wherever valid track data exists */
     float X0[OBJNUM_MAX], /* inertial frame initial positions for */
     float Y0[OBJNUM_MAX], /* each track */
     float Z0[OBJNUM_MAX],
     plural float T[3][3], /* rotation matrix from body to inertial frame
                              for each segment of all tracks */
     plural float u,       /* body frame velocity histories */
     plural float v,       /* for all tracks */
     plural float w,
     plural float *X,      /* output new inertial frame position */
     plural float *Y,      /* histories for all tracks */
     plural float *Z)
{
   int i,                  /* index counter */
       r,                  /* object counter */
       max_indx;           /* max. number of objects */
   plural float vel,       /* velocity */
                disp,      /* displacement */
                vel_temp;  /* temporary velocuty variable */
   plural float U_I,      /* inertial frame velocities */
                V_I,
                W_I;

   U_I = V_I = W_I = 0.0;
   disp = vel = 0.0;
   *X = *Y = *Z = 0.0;
   max_indx = max(n_obj[0],n_obj[1]);
   for(r=0;r<num_obj; r++){
      if(ixproc == r*3)   disp = disp + X0[r];
      if(ixproc == r*3+1) disp = disp + Y0[r];
      if(ixproc == r*3+2) disp = disp + Z0[r];
   }

   /* Converting u,v,w to U,V,W (body frame to inertial frame velocities) */
   if(len_mask){
      U_I = T[0][0]*u + T[0][1]*v + T[0][2]*w;
      V_I = T[1][0]*u + T[1][1]*v + T[1][2]*w;
      W_I = T[2][0]*u + T[2][1]*v + T[2][2]*w;
   }

   /* Forming the velocity matrix for the MPP*/
   for(r=0;r<num_obj; r++){
      if(ixproc  == 3*r) vel = xnetW[2*r].U_I;
      if(ixproc  == 3*r+1) vel = xnetW[2*r+1].V_I;
      if(ixproc  == 3*r+2) vel = xnetW[2*r+2].W_I;
   }

   /* translate disp by velocity */
   for(i=0;i<max_indx+1;i++){
      vel_temp = 0.0;
      if(iyproc == i){
         vel_temp = vel;
         xnetcS[63-i].vel_temp = vel_temp;
      }
      if(iyproc >= i) {
         disp = disp + vel_temp;
      }
   }

   /* Recovering X,Y,Z from disp matrix */
   for(r=0;r< num_obj; r++){
      if(ixproc == r){
         if(iyproc < n_obj[r]+1) {
            *X = xnetE[2*r].disp;
            *Y = xnetE[2*r+1].disp;
            *Z = xnetE[2*r+2].disp;
         }
         else {
            *X = 0.0;
            *Y = 0.0;
            *Z = 0.0;
         }
      }
   }
}

/*****************************************************************************
   function: back_conv
   This is the reverse of the main conversion function.  It converts the
   inertial frame position histories for all tracks to body frame velocity
   histories.
*****************************************************************************/
void back_conv(plural int len_mask,
               plural float X,
               plural float Y,
               plural float Z,
               float X0[OBJNUM_MAX],
                float Y0[OBJNUM_MAX],
                float Z0[OBJNUM_MAX],
                plural float T_conv[3][3],
                plural float *u,
                plural float *v,
                plural float *w)
{
   int r;
   float pi;
   plural float U,
                V,
                W;
   pi = 4.0*atan(1.0);
   U = V = W = 0.0;
   if(iyproc == 0){
      for(r = 0; r < num_obj; r++){
         if(ixproc == r){
            U = X - X0[r]; V = Y - Y0[r]; W = Z - Z0[r];
         }
      }
   }
   else if(len_mask && iyproc > 0){
      U = X - xnetN[1].X;
      V = Y - xnetN[1].Y;
      W = Z - xnetN[1].Z;
   }

   if(len_mask) {
      *u = T_conv[0][0]*U + T_conv[1][0]*V + T_conv[2][0]*W;
      *v = T_conv[0][1]*U + T_conv[1][1]*V + T_conv[2][1]*W;
      *w = T_conv[0][2]*U + T_conv[1][2]*V + T_conv[2][2]*W;
   }
   else {
      *u = *v = *w = 0.0;
   }
}

/****************************************************************************
   function: short_conv
   Converts the inertial frame position history for all tracks from
   rectangular coordinates to azimuth and elevation.
****************************************************************************/
void
short_conv(int n_obj[OBJNUM_MAX],/* length of each track (0-63) */
           plural float X,       /* inertial frame position histories */
           plural float Y,       /* for each track in rectangular */
           plural float Z,       /* coordinates */
           plural float alpha[OBJNUM_MAX/2], /* elevation angles */
           plural float beta[OBJNUM_MAX/2],  /* azimuth angles */
           plural int len_mask)
{
   int r;
   float pi;
   plural float temp_alpha,temp_beta;
   plural int address_loc;

   pi = 4.0*atan(1.0);
   *alpha = *beta = 0.0;

   /* Find the polar coordinates of location sequence  */
```

```
    /* Add pi/2 as atan finds angles from -90 to 90     */
    if(len_mask){
       if(Y < 0){
          if(X < 0){
             temp_beta = pi + (fp_atan(Y/X));
          }
          else{
             temp_beta = 2*pi + (fp_atan(Y/X));
          }
       }
       else{
          if(X < 0){
             temp_beta = pi + (fp_atan(Y/X));
          }
          else{
             temp_beta = (fp_atan(Y/X));
          }
       }
       temp_alpha = (fp_atan(Z/fp_sqrt(X*X + Y*Y)));
    }

    if(ixproc < 64){
       address_loc = nxproc * ixproc;
    }
    else{
       address_loc = ixproc *nxproc + 1;
    }

    /*** Transpose column vectors into row vectors with router command -
         The resulting plural variable alpha, beta contain angles for the
         track along the row with their copies along the columns ***/

    for(r=0;r< (num_obj/2)+(num_obj-(num_obj/2)*2);r++){
       if(iyproc == 0 &&
          (ixproc < n_obj[r]+1 || ((ixproc > 63)&&
          (ixproc < 65+n_obj[r+1])))){
          alpha[r] = router[address_loc+r*2].temp_alpha;
          xnetcS[63].alpha[r] = alpha[r];
          beta[r] = router[address_loc+r*2].temp_beta;
          xnetcS[63].beta[r] = beta[r];
       }
    }
} /* end function: short_conv */

/*************************************************************************
   function: form_sin_cos
*************************************************************************/
/* (comments by Steve Jacobs)                                            */
/* This routine takes a history of orientations and generates a         */
/* rotation matrix for each.  The plural floats theta, phi and psi      */
/* contain the pitch, roll and yaw angles, respectively, of the         */
/* 64 most recent orientations for each track (i.e. each detected       */
/* target).  The orientation history for each track is stored in a      */
/* separate column of these plural floats.  The plural int              */
/* len_mask is used to select the active set of processors as the       */
/* columns in which orientations for each track reside.  The            */
/* entries of the rotation matrices are stored in the 3x3 array of      */
/* plural floats T.  For a single orientation, the equation for T       */
/* is given by the following matrix product:                            */
/*                                                                       */
/*      T = Rx(phi) * Ry(theta) * Rz(psi)                                   */
/*                                                                          */
/*      Rx(phi) =       | 1       0          0      |                       */
/*                      | 0    cos(phi)  sin(phi)   |                       */
/*                      | 0   -sin(phi)  cos(phi)   |                       */
/*                                                                          */
/*      Ry(theta) =     | cos(theta)  0   -sin(theta)  |                    */
/*                      |     0       1        0       |                    */
/*                      | sin(theta)  0    cos(theta)  |                    */
/*                                                                          */
/*      Rz(psi) =       |  cos(psi)  sin(psi)   0 |                         */
/*                      | -sin(psi)  cos(psi)   0 |                         */
/*                      |     0          1      0 |                         */
/*                                                                          */
/***************************************************************************/

void form_sin_cos(plural int len_mask,  /* mask for active set         */
                plural float theta,     /* pitch angle history         */
                plural float phi,       /* roll angle history          */
                plural float psi,       /* yaw angle history           */
                plural float T[3][3])   /* rotation matrices           */
{

/*      The following variables provide temporary storage for the   */
/*      sines and cosines of the orientation angles.                 */

plural float cos_theta1,cos_phi1,cos_psi1;
plural float sin_theta1,sin_phi1,sin_psi1;

/*      Compute sines and cosines of orientation angles              */

cos_theta1 = fp_cos(theta);
cos_phi1 = fp_cos(phi);
cos_psi1 = fp_cos(psi);
sin_theta1 = fp_sin(theta);
sin_phi1 = fp_sin(phi);
sin_psi1 = fp_sin(psi);

/*      For the active columns, compute the entries of the           */
/*      rotation matrices.  Otherwise fill the rotation matrices     */
/*      with zeros.                                                  */

if(len_mask)
        {
        T[0][0] = cos_theta1*cos_psi1;
        T[0][1] = sin_phi1*sin_theta1*cos_psi1 - cos_phi1*sin_psi1;
        T[0][2] = cos_phi1*sin_theta1*cos_psi1 + sin_phi1*sin_psi1;
        T[1][0] = cos_theta1*sin_psi1;
        T[1][1] = sin_phi1*sin_theta1*sin_psi1 + cos_phi1*cos_psi1;
        T[1][2] = cos_phi1*sin_theta1*sin_psi1 - sin_phi1*cos_psi1;
        T[2][0] = -sin_theta1;
        T[2][1] = sin_phi1*cos_theta1;
        T[2][2] = cos_phi1*cos_theta1;
        }
else
        {
        T[0][0] =
        T[0][1] =
        T[0][2] =
        T[1][0] =
        T[1][1] =
        T[1][2] =
        T[2][0] =
        T[2][1] =
        T[2][2] = 0.0;
        }

}
```

```
/***************************************************************************
   covariance.m
****************************************************************************
   These comments last edited 6/22/95 by Jason August (jta1@cec.wustl.edu)

   The covariance function is found in this way:
   1) The state transition matrix Phi is defined as the solution of the
      matrix differential equation: dM(s)/ds=-A(theta[s], theta[s])M(s)
      where M(tau) is an identity matrix.
   2) The covariance of the body frame velocity process Kv(s1,s2) then
      becomes sigma*integral from t(0) to min(s1,s2) of:

** End Introductory Comments */

/***************************************************************************
   Header Files to be included...
***************************************************************************/
#include <stdio.h>      /* standard i/o header */
#include <math.h>       /* math header */
#include <mpl.h>        /* MasPar programming language header */
#include <mpml.h>       /* MasPar mathematics library header */
#include <su/values.h>  /* DEC header file of common values */
#include <su/stdlib.h>  /* DEC std. library.h (/usr/maspar/include/su) */
#include <head.h>       /* the big header file that everything is in */
#include <param.h>      /* system parameter header file */
/* End Header Files to be included */

/***************************************************************************
   function: form_conv
   this function forms the covariance of the position vector forming
   the track. It is parametrized by the angular velocities p_rate,
   q_rate and r_rate. The matrix is incrementally calculated by adding
   a row and a column corresponding to next time instant as the 64-window
   advances along the track.  The algorithm is structured as follows:

   K(n) = covariance matrix at time n      (3*n x 3*n)
   k(n) = last 3 columns of K(n)           (3*n x 3)
   C    = last 3 rows of k(n)              (3 x 3)

   K(n+1) = |                        |                  |
            |       K(n)             |      A*k(n)      |
            |                        |                  |
            -------------------------|------------------|
            |    ( A * k(n) )^T      | A^T * C * A^T + I |

   cov is a 192x192 block covariance matrix with 64x64 block size of
   3x3 matrices.
***************************************************************************/
void
form_cov(int n, /* time index along the track(0..63) */
             plural float cov[OBJNUM_MAX][3][2],
             int obj_ind, /* object index, track whose coavraince in
                               being calculated */
             float p_rate[OBJNUM_MAX],
             float q_rate[OBJNUM_MAX],
             float r_rate[OBJNUM_MAX]){
        int i, k, l,
        row, element;
   plural float C_temp[1][2], A,
                temp[1][2],temp_t[3][1],
             term1,term2,
             A_temp, row_temp = 0.0;
plural int address,address_loc;
float term;

/*** address for transposing temp via router ***/
address = 0;
if(ixproc < 3){
   address = ixproc * nxproc + iyproc;
}

row = 3*(n-1)/64;
element = 3*(n-1)%64;

for(i=0;i<3;i++){
   temp[0][i] = 0.0;
   C_temp[0][i] = 0.0;
}

/*** Identity for n=0 & n = 1 ***/
if(n == 0){
   proc[0][0].cov[obj_ind][0][0]  = force_var_x*1.0;
   proc[1][1].cov[obj_ind][0][0]  = force_var_y*1.0;
   proc[2][2].cov[obj_ind][0][0]  = force_var_z*1.0;
}
else if(n == 1){
   proc[3][3].cov[obj_ind][0][0]  = force_var_x*1.0;
   proc[4][4].cov[obj_ind][0][0]  = force_var_y*1.0;
   proc[5][5].cov[obj_ind][0][0]  = force_var_z*1.0;
}
else{
   /*** Form A matrix for this n ***/
   /*** A is the skew-symmetric matrix of angular velocities ***/
   /***  lambda corresponds to the drag coefficient ****/
   A = 0.0;
   l = obj_ind;
   proc[0][1].A = -r_rate[l];proc[0][2].A = q_rate[l];
   proc[1][0].A = r_rate[l]; proc[1][2].A = -p_rate[l];
   proc[2][0].A = -q_rate[l];proc[2][1].A = p_rate[l];
   if(ixproc == iyproc && iyproc <3){
      A = 1.0 - lambda;
   }
   A_temp = A;
   address_loc  = ixproc * nxproc + iyproc;
   if(ixproc < 3 && iyproc < 3){
      A = router[address_loc].A;
   }

   /** Special case when n == 22 or 43 because the covariance blocks
   span across the plural variables **/
   /*** Form the temp matrix to multiply with A ***/
   if(n == 22 || n ==43){
      if(n == 22){
         for(i=0;i<2;i++){
            if(iyproc == 0){
               C_temp[0][i] = xnetS[63].cov[obj_ind][0][i];
            }
            if(iyproc == 1){
               C_temp[0][i] = xnetN[1].cov[obj_ind][1][i];
            }
         if(iyproc == 2){
            C_temp[0][i] = xnetN[1].cov[obj_ind][1][i];
         }
      }
   }
   if(n == 43){
      for(i=0;i<2;i++){
         if(iyproc == 0){
            C_temp[0][i] = xnetS[62].cov[obj_ind][1][i];
         }
         if(iyproc == 1){
            C_temp[0][i] = xnetS[62].cov[obj_ind][1][i];
         }
         if(iyproc == 2){
            C_temp[0][i] = xnetN[2].cov[obj_ind][2][i];
         }
      }
   } /* end if(n=43...) */

else{
  /*** Grab the last 3 columns from the existing covariance matrix ***/
   C_temp[0][0] = fp_matblkget(&cov[obj_ind][row][0],128,128,element,0);
   C_temp[0][1] = fp_matblkget(&cov[obj_ind][row][1],128,128,element,0);
}

  /*** Multiply those 3 columns with the 3x3 A matrix ***/
fp_matmul(&temp[0][0],&A,&C_temp[0][0],3,3,256);

  /** Transpose the resulting columns to get 3  row vectors **/
for(i=0;i<3;i++){
   temp_t[i][0] = 0.0;
}

if(ixproc < 3){
   temp_t[0][0] = router[address].temp[0][0];
   temp_t[1][0] = router[address+64].temp[0][0];
   temp_t[2][0] = router[address].temp[0][1];
}

  /** Subsitiue back the new row and column in the covariance
  matrix corresponding to the next time instant ***/
for(i=0;i<2;i++){
   if(iyproc == ((3*n)%64)){
      cov[obj_ind][(3*n)/64][i] = xnetN[(3*n)%64].temp[0][i];
   }
   if(iyproc == ((3*n+1)%64)){
      cov[obj_ind][(3*n+1)/64][i] = xnetN[(3*n)%64].temp[0][i];
   }
   if(iyproc == ((3*n+2)%64)){
      cov[obj_ind][(3*n+2)/64][i] = xnetN[(3*n)%64].temp[0][i]
   }
}

if(n != 42){
   fp_matcopy(192,3,&temp_t[0][0],128,0,0,&cov[obj_ind][0][0],256,0,3*n);
}
else{
   for(i=0;i<3;i++){
      if(ixproc == 126){
         cov[obj_ind][i][0] = xnetW[126].temp_t[i][0];
```

```
            }
            if(ixproc == 127){
               cov[obj_ind][i][0] = xnetW[126].temp_t[i][0];
            }
            if(ixproc == 0){
               cov[obj_ind][i][1] = xnetE[2].temp_t[i][0];
            }
         }
      }

      /** Derive the 3x3 covariance matrix of the last track segment **/
      /*** obtain the bottom 3x3 matrix at n-1 column ***/
      term1 = term2 = 0.0;
      if(element == 62 || element == 63){
         if(element == 63){
            term1 = fp_matblkget(&temp_t[row][0],64,128,element,0);
            if(iyproc == 1){
               term1 = xnetN[1].temp_t[1][0];
            }
            if(iyproc == 2){
               term1 = xnetN[1].temp_t[1][0];
            }
         }

         if(element == 62){
            term1 = fp_matblkget(&temp_t[row][0],64,128,element,0);
            if(iyproc == 2){
               term1 = xnetN[2].temp_t[2][0];
            }
         }
      }
      else{
         term1 = fp_matblkget(&temp_t[row][0],64,128,element,0);
      }

      fp_matmul(&term2,&A_temp,&term1,3,3,3);
      proc[0][0].term2 = proc[0][0].term2 + force_var_x*1.0;
      proc[1][1].term2 = proc[1][1].term2 + force_var_y*1.0;
      proc[2][2].term2 = proc[2][2].term2 + force_var_z*1.0;

      /** Substitute the 3x3 covariance matrix for the last segment in
         the covariance matrix **/
      fp_matcopy(3,3,&term2,64,0,0,&cov[obj_ind][0][0],256,3*n,3*n);
   }
} /* end function: form_conv */

/*****************************************************************************
   function: shift_conv_in
   shifts the covariance martix towards the upper left corner as new
   data replaces older data.
*****************************************************************************/
void
shift_cov_in(int obj_ind, plural float A[OBJNUM_MAX][3][2])
{
   int i,j;

   /*** shift left ***/
   for(i=0;i< 3; i++){
      for(j=0;j<2;j++){
         A[obj_ind][i][j] = xnetE[3].A[obj_ind][i][j];
```

```
            if(ixproc > 124){
               if(j == 0){
                  A[obj_ind][i][j] = xnetW[125].A[obj_ind][i][j+1];
               }
               else{
                  A[obj_ind][i][1] = 0.0;}
               }
            }
        }
        /*** shift up ***/
        for(j=0;j<2;j++){
           for(i=0;i<3;i++){
              A[obj_ind][i][j] = xnetS[3].A[obj_ind][i][j];
              if(iyproc > 60){
                 if(i< 2)
                    A[obj_ind][i][j] = xnetN[61].A[obj_ind][i+1][j];
                 else
                    A[obj_ind][2][j] = 0.0;
              }
           }
        }
} /* end function: shift_conv_in */

/***************************************************************************
   function: angle_diff
   finds the difference between two angles (x1 and x2)
***************************************************************************/
void
angle_diff(float x1,float x2,float *y)
{
   register float t,a,b,pi;
   register float x1_abs,x2_abs;
   float term;

   pi = 4.0*atan(1.0);

   if(x1 < 0){
      x1_abs = x1 + 2*pi;
   }
   else{
      x1_abs = x1;
   }
   if(x2 < 0){
      x2_abs = x2 + 2*pi;
   }
   else{
      x2_abs = x2;
   }


   if(x1_abs > x2_abs){
      t = x1_abs;
   }
   else{
      t = x2_abs;
   }

   a = fabs(2*pi+x1_abs+x2_abs-2*t);
   b = fabs(x1_abs-x2_abs);
   if(a < b){
        *y  = 2*pi+x1_abs+x2_abs-2*t;
    }
    else{
        *y = x1_abs-x2_abs;
    }
} /* end function: angle_diff */
```

```
/***************************************************************************
   data.m
****************************************************************************
   These comments last edited 6/26/95 by Jason August (jta1@cec.wustl.edu)

   This program takes the inertial frame rectangular locations of each
   of the targets present in the scene at that time and computes the
   complex data vector generated by the cross array. This involves
   calculating the direction vector of each target and adding them up and
   finally adding the noise to the result.
** End Introductory Comments */

/***************************************************************************
   Header Files to be included...
**************************************************************************/
#include <stdio.h>      /* standard i/o header */
#include <math.h>       /* math header */
#include <mpl.h>        /* MasPar programming language header */
#include <mpml.h>       /* MasPar mathematics library header */
#include <head.h>       /* the big header file that everything is in */
#include <param.h>      /* system parameter header file */
/* End Header Files to be included */

/***************************************************************************
 Global Variable Declarations
**************************************************************************/
extern plural float index_u,
                    index_d;
extern float track[OBJNUM_MAX][tracksize][3];
extern int num_obj;

/***************************************************************************
   function: data
   ...takes intertial frame rectangular locations of targets and computes
   the complex data vector generated by the cross array.  In other words,
   it takes data gathered from the objects and transforms it so that the
   locations can be displayed on the SGI.
**************************************************************************/
void
data(int len_obj[OBJNUM_MAX],
     int max_indx,
     plural float *d_r,
     plural float *d_i)
{
   int r,                                 /* index variable */
       max_len;                           /* length of longest track */
   plural float rand_r, rand_i;           /* random number matricies */
   register plural float alpha, beta,     /* holds azimuth-elevation info */
                         temp, dr, di;
   float pi,
         alpha_temp,beta_temp,            /* temporary variables used to */
         X_temp,Y_temp,Z_temp;            /* speed up calculations */

        printf("DATA: starting.....\n");
   pi = 4.0*atan(1.0);
   di = dr = 0.0;

        printf("DATA: finding longest track\n");
   /* Find longest track */
   max_len = -1;
 for(r=0;r<num_obj;r++){
    max_len = max(max_len,len_obj[r]);
 }

 for(r=0; r < OBJNUM_TRU; r++){
              printf("DATA: start estimating\n");
    /*** Start estimating for track[1][] ***/
    X_temp = track[r][max_len+1][0];
    Y_temp = track[r][max_len+1][1];
    Z_temp = track[r][max_len+1][2];

              printf("DATA: find AZ; EL\n");
              printf("X_temp = %f, Y_temp = %f \n",X_temp,Y_temp);
    /*** Find azimuth-elevation coordinates ***/
    beta_temp = (atan(Y_temp/X_temp));
              printf("DATA: find AZ, EL 1\n");
    alpha_temp = (atan(Z_temp/sqrt(X_temp*X_temp + Y_temp*Y_temp)));
              printf("DATA: find AZ, EL 2\n");
    if(Y_temp < 0){
       if(X_temp < 0){
          beta = beta_temp + pi;
              printf("DATA: find AZ, EL 3\n");
       }
       else{
          beta = beta_temp + 2*pi;
              printf("DATA: find AZ, EL 4\n");
       }
    }
    else{
       if(X_temp < 0){
          beta = beta_temp + pi;
              printf("DATA: find AZ, EL 5\n");
       }
       else{
          beta = beta_temp;
              printf("DATA: find AZ, EL 6\n");
       }
    }
              printf("DATA: find AZ, EL 7\n");
    alpha = alpha_temp;

              printf("DATA: form direction vectors \n");
/*** Form direction vectors for each target ***/
    if(ixproc == max_indx || ixproc == max_indx+64){
       temp = pi*(index_d* fp_cos(alpha)*fp_sin(beta)
              + index_u * fp_cos(alpha)*fp_cos(beta));
       dr = dr + fp_cos(temp);
       di = di - fp_sin(temp);
    }
    else{
       dr = di = 0.0;
    }
 }

              printf("DATA: generate noise \n");
 /*** Generate noise ***/
 GaussRand(&rand_r);
 GaussRand(&rand_i);
```

```
                printf("DATA:  final result, including noise \n");
   /* Final result, including noise... */
   if(ixproc == max_indx || ixproc == max_indx+64){
      *d_r = (S_R*dr - S_I*di) + 0*N_R*rand_r;
      *d_i = (S_R*di + S_I*dr) + 0*N_I*rand_i;
   }
} /* end function: data */

/*************************************************************************
   function: range_data
   Generates the range data for each target.  Computes true range
   for each track segment from true X,Y,Z values.
*************************************************************************/
void
range_data(int index[OBJNUM_MAX],
           int len_obj[OBJNUM_MAX],
           int max_indx,
           plural float *range_dat)
{
   int r, max_len;
   float X_temp,Y_temp,Z_temp;

   /* Find the longest track */
   max_len = -1;
   for(r=0;r<num_obj;r++){
      max_len = max(max_len,len_obj[r]);
   }

   for(r=0;r<num_obj;r++) {
      X_temp = track[index[r]][max_len+1][0];
      Y_temp = track[index[r]][max_len+1][1];
      Z_temp = track[index[r]][max_len+1][2];

      /*** Forming the range data ***/
      if(iyproc == max_indx && ixproc == r){
         *range_dat = sqrt(X_temp*X_temp + Y_temp*Y_temp + Z_temp*Z_temp);
      }
   }
} /* end function: range_data */
```

```
/***************************************************************************
   diff.m
****************************************************************************
   These comments last edited 6/26/95

   Diff.m performs the gradient cycles on the log posterior energy (log
   likelihood + log prior) with respect to the parameters X, Y, Z at each
   time (on the 64 length window) for each track being estimated.

   Since the data log likelihood L is a function of
   the azimuth elevation angles, the gradient of log likelihood involves a
   chain rule (derivative wrt XYZ = [derivative likelihood  wrt alpha, beta]
    * [derivative of alpha,beta wrt XYZ] + [derivative likelihood wrt
   range] * [derivative of range wrt XYZ]). (wrt = with respect to)

   Finally, the parameter vector gets updated:
   X(k+1) = X(k) + gradient posterior energy * dt + sqrt{dt}*wiener process
   Instead for faster implemetation we use
   X(k+1) = X(k) + sqrt-prior-covariance * gradient-likelihood *dt
                 + sqrt{dt}*wiener process
** End Introductory Comments */

/***************************************************************************
   Header Files to be included...
***************************************************************************/
#include <stdio.h>      /* standard i/o header */
#include <math.h>       /* math header */
#include <mpl.h>        /* MasPar programming language header */
#include <mpml.h>       /* MasPar mathematics library header */
#include <head.h>       /* the big header file that most everything is in */
#include <head2.h>      /* the big header file that everything else is in */
#include <param.h>      /* system parameter header file */
/* End Header Files to be included */

/***************************************************************************
 Global Variable Declarations
***************************************************************************/
extern plural float index_u,
                    index_d;
extern int outsock, /* pointer to outgoing socket information */
           num_obj; /* the estimated number of objects in the scene */
extern float track[OBJNUM_MAX][tracksize][3],
             pitch[OBJNUM_MAX][tracksize],
             yaw[OBJNUM_MAX][tracksize],
             roll[OBJNUM_MAX][tracksize];


/***************************************************************************
   diff function
***************************************************************************/
void
diff(int iter,               /* */
     int sel,                /* selects the object that's being considered */
     int n_obj[OBJNUM_MAX],  /* the track length, 0..63 (most recent 64) */
     int len_obj[OBJNUM_MAX], /* the true (full) track length */
     plural float dat_r, plural float dat_i,
     plural float range_dat,
     plural float *X, plural float *Y, plural float *Z,
     float X0[OBJNUM_MAX], float Y0[OBJNUM_MAX], float Z0[OBJNUM_MAX],
     plural float cov_out[OBJNUM_MAX][3][2],
        plural float C_x[OBJNUM_MAX][3][2],
        plural int len_mask)
{
    /* variable delarations */
    float pi;
    register int d,l,r,max_indx;
    plural int address_loc;
    plural float dt_X, dt_Y, dt_Z,
                 X_temp, Y_temp, Z_temp,
                 der_alpha_r,der_alpha_i,
                 der_beta_r,der_beta_i,
                 dr,di,s_d_r,s_d_i,
                 range1,range2,range,
                 cos_alpha[OBJNUM_MAX/2],cos_beta[OBJNUM_MAX/2],
                 sin_alpha[OBJNUM_MAX/2],sin_beta[OBJNUM_MAX/2],
                 s_db_r,s_db_i,s_da_r,s_da_i,
                 der_r_X,der_r_Y,der_r_Z,der_liker_r,
                 der_liker_X,der_liker_Y,der_liker_Z,
                 dX[3],
                 alpha[OBJNUM_MAX/2], /* azimuth angles from track */
                 beta[OBJNUM_MAX/2];  /* elevation angles from track */

    register plural float der_like_alpha, der_like_beta,
                          der_alpha_X, der_alpha_Y, der_alpha_Z,
                          der_beta_X, der_beta_Y, der_beta_Z,
                          der_like_X, der_like_Y, der_like_Z,
                          term1, term2,
                          temp_alpha, temp_beta,
                          sin_cosa_cosb,sin_cosa_sinb,
                          cos_cosa_cosb,cos_cosa_sinb,
                          p_i_cosa_sinb,p_i1_cosa_cosb,
                          p_i_sina_cosb,p_i1_sina_sinb,
                          p_i_cosa_cosb,p_i1_cosa_sinb;

    /* variable initialization */
    pi = 4.0*atan(1.0);
    X_temp = *X; Y_temp = *Y; Z_temp = *Z;
    range1 = 0.0;
    der_liker_X = der_liker_Y = der_liker_Z = 0.0;
    der_like_X = der_like_Y = der_like_Z = 0.0;
    temp_alpha = temp_beta = 0.0;
    max_indx = 0;

    /* find maximum full track length */
    for(r=0;r<num_obj;r++){
       max_indx = max(max_indx,len_obj[r]);
    }

    /* initialization of step sizes for diffusion */

    /* if the full track length fits in a window compute step sizes
       from position history */
    if(max_indx < 63){
       if(len_mask){
          dt_X = 0.2*(1.2/((max_indx+1)*(max_indx+1)))*(p_fabs(X_temp/1000.0));
          dt_Y = 0.2*(1.2/((max_indx+1)*(max_indx+1)))*(p_fabs(Y_temp/1000.0));
          dt_Z = 0.2*(1.2/((max_indx+1)*(max_indx+1)))*(p_fabs(Z_temp/1000.0));
               }
       else{
          dt_X = dt_Y =  dt_Z = 0.0;
```

```
   }
     }

/* if the full track length is to big to fit in the 64 length window
   then the step size is constant (0.00012) where len_mask is true
   and 0 where len_mask is false  */
else{
   if(len_mask){
      dt_X =
      dt_Y =
      dt_Z = 0.00012; /* 1.2 * 0.0001 */
             }
   else{
      dt_X = dt_Y = dt_Z = 0.0;
   }
}

for(l=0;l<iter;l++){
   dr = di = s_d_r = s_d_i = 0.0;
   short_conv(n_obj,X_temp,Y_temp,Z_temp,alpha,beta,len_mask);

/* compute projection of data vector into directions given by
   alpha, beta values from track. */

   for(r=0;r<num_obj/2 + (num_obj-(num_obj/2)*2);r++){
      if(ixproc < n_obj[2*r]+1 || (ixproc > 63 && ixproc < n_obj[2*r+1]+65)){
         cos_alpha[r] = fp_cos(alpha[r]);
         cos_beta[r]  = fp_cos(beta[r]);
         sin_alpha[r] = fp_sin(alpha[r]);
         sin_beta[r]  = fp_sin(beta[r]);

         /* Calculate terms once */
         p_i_cosa_cosb = pi*index_u*cos_alpha[r]*cos_beta[r];
         p_i_cosa_sinb  = pi*index_u*cos_alpha[r]*sin_beta[r];
         p_i_sina_cosb  = pi*index_u*sin_alpha[r]*cos_beta[r];
         p_i1_sina_sinb = pi*index_d*sin_alpha[r]*sin_beta[r];
         p_i1_cosa_sinb = pi*index_d*cos_alpha[r]*sin_beta[r];
         p_i1_cosa_cosb = pi*index_d*cos_alpha[r]*cos_beta[r];

         sin_cosa_cosb =  fp_sin(p_i_cosa_cosb);
         sin_cosa_sinb =  fp_sin(p_i1_cosa_sinb);
         cos_cosa_sinb =  fp_cos(p_i1_cosa_sinb);
         cos_cosa_cosb =  fp_cos(p_i_cosa_cosb);

         dr = fp_cos(p_i1_cosa_sinb + p_i_cosa_cosb);
         di = -fp_sin(p_i1_cosa_sinb + p_i_cosa_cosb);
      }
      else{
         dr = di = 0.0;
      }

      s_d_r += (S_R*dr-S_I*di);
      s_d_i += (S_R*di+S_I*dr);
     }

   if(ixproc < 64){
      s_d_r += xnetE[64].s_d_r;
      s_d_i += xnetE[64].s_d_i;
   }
   if(ixproc > 63){
      s_d_r = xnetW[64].s_d_r;
      s_d_i = xnetW[64].s_d_i;
   }

   /* form the log-likelihood derivative with respect to alpha and beta;
      take the derivative of the direction vector wrt alpha and beta.
      der = derivative */

   for(r=0;r<num_obj/2 + (num_obj-(num_obj/2)*2);r++){
      if(ixproc < n_obj[2*r]+1 ||
         (ixproc > 63 && ixproc < n_obj[2*r+1]+65)){
         der_beta_r = p_i_cosa_sinb*sin_cosa_cosb -
                      p_i1_cosa_cosb*sin_cosa_sinb;

         der_beta_i = p_i_cosa_sinb*cos_cosa_cosb -
                      p_i1_cosa_cosb*cos_cosa_sinb;

         der_alpha_r = p_i_sina_cosb*sin_cosa_cosb +
                       p_i1_sina_sinb*sin_cosa_sinb;

         der_alpha_i = p_i_sina_cosb * cos_cosa_cosb +
                       p_i1_sina_sinb * cos_cosa_sinb;

         /* Calculate s der_beta products */
         s_db_r = S_R*der_beta_r - S_I*der_beta_i;
         s_db_i = S_R*der_beta_i + S_I*der_beta_r;
         s_da_r = S_R*der_alpha_r - S_I*der_alpha_i;
         s_da_i = S_R*der_alpha_i + S_I*der_alpha_r;

         term1 = (dat_r - s_d_r) * s_db_r;
         term2 = (dat_i - s_d_i) * s_db_i;
         temp_beta = 2*(term1 +term2)/(N_R*N_R+N_I*N_I);

         term1 = (dat_r - s_d_r) * s_da_r;
         term2 = (dat_i - s_d_i) * s_da_i;
         temp_alpha = 2*(term1 +term2)/(N_R*N_R+N_I*N_I);

         term1 = 0.0;
         term2  = 0.0;

/* sum derivatives over direction vector index */

         for (d=1; d<nyproc; d = d<<1) /* then results go south */
            if ((iyproc & d-1) == d-1)
               temp_alpha += xnetpN[d].temp_alpha;
             term1 = temp_alpha;

             for (d=1; d<nyproc; d = d<<1) /* then results go south */
            if ((iyproc & d-1) == d-1)
               temp_beta += xnetpN[d].temp_beta;
         term2 = temp_beta;

         if(iyproc == 63){
            xnetcN[63].term1 = term1;
            xnetcN[63].term2 = term2;
         }
      }

/* derivative of likelihood wrt alpha = transpose(term1)
derivative of likelihood wrt beta = transpose(term2) */

   address_loc = iyproc;
   if(ixproc == 2*r && iyproc < n_obj[2*r]+1){
      der_like_alpha = router[iyproc].term1;
      der_like_beta = router[iyproc].term2;
   }

   if(ixproc == 2*r+1 && iyproc < n_obj[2*r+1]+1){
      der_like_alpha = router[iyproc+64].term1;
      der_like_beta = router[iyproc+64].term2;
   }
}

  dX[0] = dX[1] = dX[2] = 0.0;

if(len_mask){

/* Compute ranges */

   range1 = (X_temp*X_temp + Y_temp*Y_temp + Z_temp*Z_temp);
   range2 = fp_sqrt(X_temp*X_temp + Y_temp*Y_temp);

/* Find derivatives of angles alpha & beta wrt displacements X,Y,Z */

   der_alpha_X = -X_temp*Z_temp/(range1*range2);
   der_beta_X  = -Y_temp/(range2*range2);
   der_alpha_Y = -Y_temp*Z_temp/(range1*range2);
   der_beta_Y  = X_temp/(range2*range2);
   der_alpha_Z = range2/range1;

/* First term in derivative of like wrt XYZ
   [der like alpha,beta] * [der alpha,beta wrt XYZ] */

   der_like_X = (der_like_alpha*der_alpha_X + der_like_beta*der_beta_X);
   der_like_Y = (der_like_alpha*der_alpha_Y + der_like_beta*der_beta_Y);
   der_like_Z = (der_like_alpha*der_alpha_Z);

/* Find derivative of the range data likelihood wrt parameters X, Y, Z */

   range = fp_sqrt(range1);
   der_r_X = X_temp/range;
   der_r_Y = Y_temp/range;
   der_r_Z = Z_temp/range;

/* Second term in derviative of like wrt XYZ
   [der like wrt range] * [der range wrt XYZ] */

   der_liker_r = (range_dat- range)/(ran_var*ran_var);

   der_liker_X = der_liker_r*der_r_X;
   der_liker_Y = der_liker_r*der_r_Y;
   der_liker_Z = der_liker_r*der_r_Z;

/* Add two derivative terms */

   dX[0] = len_mask*(der_like_X + der_liker_X);
   dX[1] = len_mask*(der_like_Y + der_liker_Y);
   dX[2] = len_mask*(der_like_Z + der_liker_Z);
}
```

```
        /* Multiply likelihood derivative by
           sqrt(tracking_data_covariance) */

        mymatvecmul(n_obj, dX, cov_out);

        /* position update = posterior derivative * step size */

        if(len_mask){
           X_temp = X_temp + dX[0]*dt_X;
           Y_temp = Y_temp + dX[1]*dt_Y;
           Z_temp = Z_temp + dX[2]*dt_Z;

           if(Z_temp < 0.0)
              Z_temp = 0.0;
        }
     }

     if(len_mask){
        *X = X_temp; *Y = Y_temp; *Z = Z_temp;
     }

} /* end function diff */
```

```
/***************************************************************************
  initialization.m
****************************************************************************
  These comments last edited 6/26/95 by Jason August (jtal@cec.wustl.edu)
** End Introductory Comments */

/***************************************************************************
  Header Files to be included...
***************************************************************************/
#include <stdio.h>      /* standard i/o header */
#include <math.h>       /* math header */
#include <mpl.h>        /* MasPar programming language header */
#include <mpml.h>       /* MasPar mathematics library header */
#include <ppeio.h>      /* parallel processing environment i/o header */
#include <fcntl.h>      /* file system control header */
#include <sys/file.h>
#include <su/values.h>  /* DEC header file of common values */
#include <su/stdlib.h>  /* DEC std. library.h (/usr/maspar/include/su) */
#include <su/unistd.h>
#include <head.h>       /* the big header file that everything is in */
#include <param.h>      /* system parameter header file */
/* End Header Files to be included */

/***************************************************************************
  Global Variable Declarations
***************************************************************************/

        extern  int send_to_jupiter_1995;
        extern  int send_to_jupiter_1991;
        extern  int recv_from_jupiter_1992;
        extern  float track[OBJNUM_MAX][tracksize][3],pitch[OBJNUM_MAX][tracksize],
                        yaw[OBJNUM_MAX][tracksize], roll[OBJNUM_MAX][tracksize];

        extern int num_obj;

        visible extern int sendinit_jupiter();
        visible extern int recvinit_p();

        extern int do_jupiter;

/***************************************************************************
  initialization function
   initializes socket for tracking display, and then sends true track
   accross new socket.
***************************************************************************/
void init_jupiter()
{
                plural unsigned char tempimage[1024];
                int num, i,fd, flag = 1;
        plural int number;
        char name[80];
        float  orient[5544][3];


/******  Initialize socket to SGI for display ******/

if (do_jupiter) {
        send_to_jupiter_1995 = callRequest(sendinit_jupiter,4,1995);
            if (send_to_jupiter_1995 == -1){
                printf("error socket 1\n");
                exit(1); }
}

/* Send true track data to SGI */

if (do_jupiter) {
          write(send_to_jupiter_1995,&flag,sizeof(int));
          write(send_to_jupiter_1995,&track[0][0][0],
                                         tracksize*OBJNUM_TRU*3*sizeof(float));
          write(send_to_jupiter_1995,&pitch[0][0],
                                         OBJNUM_TRU*tracksize*sizeof(float));
          write(send_to_jupiter_1995,&yaw[0][0],
                                         OBJNUM_TRU*tracksize*sizeof(float));
          write(send_to_jupiter_1995,&roll[0][0],
                                         OBJNUM_TRU*tracksize*sizeof(float));
                }

printf("Init: Paths sent to jupiter for display\n");

}
```

```
/**************************************************************************
  initialization.m
***************************************************************************
  These comments last edited 6/26/95 by Jason August (jtal@cec.wustl.edu)
** End Introductory Comments */

/**************************************************************************
  Header Files to be included...
*************************************************************************/
#include <stdio.h>      /* standard i/o header */
#include <math.h>       /* math header */
#include <mpl.h>        /* MasPar programming language header */
#include <mpml.h>       /* MasPar mathematics library header */
#include <ppeio.h>      /* parallel processing environment i/o header */
#include <fcntl.h>      /* file system control header */
#include <sys/file.h>
#include <su/values.h>  /* DEC header file of common values */
#include <su/stdlib.h>  /* DEC std. library.h (/usr/maspar/include/su) */
#include <su/unistd.h>
#include <head.h>       /* the big header file that everything is in */
#include <param.h>      /* system parameter header file */
/* End Header Files to be included */

/**************************************************************************
  Global Variable Declarations
*************************************************************************/

        extern  int send_to_jupiter_1995;
        extern  int send_to_jupiter_1991;
        extern  int recv_from_jupiter_1992;
        extern  float track[OBJNUM_MAX][tracksize][3],pitch[OBJNUM_MAX][tracksize],
                        yaw[OBJNUM_MAX][tracksize], roll[OBJNUM_MAX][tracksize];

        extern int num_obj;

        visible extern int sendinit_Rome_Iris();
        visible extern int recvinit_p();

        extern int do_jupiter;

/**************************************************************************
  initialization function
   initializes socket for tracking display, and then sends true track
   accross new socket.
*************************************************************************/
void initialization()
{
                plural unsigned char tempimage[1024];
                int num, i,fd, flag = 1;
        plural int number;
        char name[80];
        float  orient[5544][3];


/******  Initialize socket to SGI for display ******/

if (do_jupiter) {
        send_to_jupiter_1995 = callRequest(sendinit_Rome_Iris,4,1995);
            if (send_to_jupiter_1995 == -1){
                printf("error socket 1\n");
                exit(1); }
}

/* Send true track data to SGI */

if (do_jupiter) {
          write(send_to_jupiter_1995,&flag,sizeof(int));
          write(send_to_jupiter_1995,&track[0][0][0],
          tracksize*OBJNUM_TRU*3*sizeof(float));
          write(send_to_jupiter_1995,&pitch[0][0],
                                                                  OBJNUM_TRU*tracksize*sizeof(float)
);
          write(send_to_jupiter_1995,&yaw[0][0],
                                                                  OBJNUM_TRU*tracksize*sizeof(float)
);
          write(send_to_jupiter_1995,&roll[0][0],
                                                                  OBJNUM_TRU*tracksize*sizeof(float)
);
                }

        printf("Init: Paths sent to jupiter for display\n");

}
```

```
/***************************************************************************
   jump_process.m
****************************************************************************
   These comments last edited 6/26/95

   Jump_Process caluclates the body frame velocity candidate for the
   track birth option.

   It uses the discrete version of the equations of motion to sample from the
   prior. A sample from vector Gausssian process is substitued for the forces
   driving the equation and the equations are solved for v[n+1] using:

   v[n+1] = v[n](I-Q[n]) + f[n]

   where Q[n] is the skew-symmetric matrix of angular veclocities
   p_rate, q_rate, r_rate

   Using the orientations pitch, yaw and roll, the program calculates
   angular velocities, and then forms the matrix Q to evaluate v[n+1] from
   the above equation

   if tracklength = 0 then v[n+1] = 0.
** End Introductory Comments */

/***************************************************************************
   Header Files to be included...
***************************************************************************/
#include <stdio.h>      /* standard i/o header */
#include <math.h>       /* math header */
#include <mpl.h>        /* MasPar programming language header */
#include <mpml.h>       /* MasPar mathematics library header */
#include <head.h>       /* the big header file that everything is in */
#include <param.h>      /* system parameter header file */
/* End Header Files to be included */

/***************************************************************************
 External Variable Declarations
***************************************************************************/
extern plural float JumpRand;

/***************************************************************************
   jump_metro function (see above comments)
***************************************************************************/
void
jump_metro(int n_obj[OBJNUM_MAX], int sel, int read_indx,
           plural float theta, plural float phi, plural float psi,
           plural float *u, plural float *v, plural float *w,
           plural float T[3][3], float p_rate[OBJNUM_MAX],
           float q_rate[OBJNUM_MAX], float r_rate[OBJNUM_MAX])
{
   /* variable declaration */
   int               r;
   float             u_term, v_term, w_term,
                     u_n[OBJNUM_MAX], v_n[OBJNUM_MAX], w_n[OBJNUM_MAX],
                     pitch, roll, yaw,
                     theta_p, phi_p, psi_p;
   register float    term,
                     cos_theta, cos_phi, cos_psi,
                     sin_theta, sin_phi, sin_psi;
```

```
/* if no track segments velocities = 0 */

if (n_obj[sel] == 0)
{
   u_n[sel] = 0;
   v_n[sel] = 0;
   w_n[sel] = 0;
}

else
{
   pitch = proc[n_obj[sel] - 1][sel].theta;
   roll = proc[n_obj[sel] - 1][sel].phi;
   yaw = proc[n_obj[sel] - 1][sel].psi;

   u_term = proc[n_obj[sel] - 1][sel].* u;
   v_term = proc[n_obj[sel] - 1][sel].* v;
   w_term = proc[n_obj[sel] - 1][sel].* w;

/* if only 1 track segment, angular velocities = 0 */

   if (n_obj[sel] == 1)
   {
      theta_p = 0.0;
      psi_p = 0.0;
      phi_p = 0.0;
   }

/* otherwise, compute change in orientation from last segment */

   else
   {
      term = proc[n_obj[sel] - 2][sel].theta;
      angle_diff(term, pitch, &theta_p);
      term = proc[n_obj[sel] - 2][sel].phi;
      angle_diff(term, roll, &phi_p);
      term = proc[n_obj[sel] - 2][sel].psi;
      angle_diff(term, yaw, &psi_p);
   }

   cos_theta = cos(pitch);
   sin_theta = sin(pitch);
   sin_phi = sin(roll);
   cos_phi = cos(roll);

   /* final calculation of angular velocities */
   p_rate[sel] = phi_p - psi_p * sin_theta;
   q_rate[sel] = theta_p * cos_phi - psi_p * cos_theta * sin_phi;
   r_rate[sel] = -theta_p * sin_phi + psi_p * cos_theta * cos_phi;

   /* solve:  v[n+1] = (I-Q[n]) v[n] */
   u_n[sel] = u_term + r_rate[sel] * v_term - q_rate[sel] * w_term;
   v_n[sel] = -r_rate[sel] * u_term + v_term + p_rate[sel] * u_term;
   w_n[sel] = q_rate[sel] * u_term - p_rate[sel] * v_term + w_term;
}

/* Gaussian driving function */
if (read_indx == 0)
   GaussRand(&JumpRand);
```

```
    /* Constructing the candidate vector of velocities */
    /**** Add f[n] *****/
    if (iyproc == n_obj[sel] && ixproc == sel)
    {
       *u=u_n[sel]+0.1*sqrt(force_var_x)*proc[4096*sel+3*read_indx].JumpRand;
       *v=v_n[sel]+0.1*sqrt(force_var_y)*proc[4096*sel+3*read_indx+1].JumpRand;
       *w=w_n[sel]+0.1*sqrt(force_var_z)*proc[4096*sel+3*read_indx+2].JumpRand;
    }
}
```

```
/**************************************************************************
   like_calculation.m
***************************************************************************
   These comments last edited 6/28/95

   The following functions compute the likelihood of various options
   (the linear Kalman filter state equations for tracking, guiding, or
   providing initial conditions for the gradient based estimators are
   generated from the likelihood)
** End Introductory Comments */

/**************************************************************************
   Header Files to be included...
**************************************************************************/
#include <stdio.h>      /* standard i/o header */
#include <math.h>       /* math header */
#include <mpl.h>        /* MasPar programming language header */
#include <mpml.h>       /* MasPar mathematics library header */
#include <head.h>       /* the big header file that everything is in */
#include <param.h>      /* system parameter header file */
/* End Header Files to be included */

/**************************************************************************
 Global Variable Declarations
**************************************************************************/
extern plural float index_u,
                    index_d;
extern int num_obj;            /* number of objects in scene */

/**************************************************************************
   likelihood function
   Calculates the probability for an option - segment birth/death or track
      birth
**************************************************************************/
void
like(int option, int sel, int n_obj[OBJNUM_MAX], plural float dat_r,
                plural float dat_i, plural float range_dat,
                plural float alpha[OBJNUM_MAX/2],
                plural float beta[OBJNUM_MAX/2], float *likelihood,plural float X,
                plural float Y, plural float Z)
{
   /* variable declaration */
   int r,
       max_indx;
   float pi,
         term;
   register plural float dr, di,
                         temp1, temp2,
                         cos_alpha, cos_beta,
                         sin_alpha, sin_beta;
   plural int len_mask;
   plural float temp3,
                dat_r_loc, dat_i_loc,
                Sig_r, Sig_i,
                temp,
                range_temp1, range_temp2;

   /* initialize variables */
   pi = 4.0*atan(1.0);
   Sig_r = Sig_i= 0.0;
```

```
dr = di = 0.0;

/* find the longest track window (window = n_obj = 0..63) */
max_indx = -1;
for(r=0;r<num_obj;r++){
   max_indx = max(max_indx,n_obj[r]);
}

/* Select track length according to option */
if(option == 1) /* delete the track segment */
   n_obj[sel] = n_obj[sel]-2;
if(option == 2) /* Not deletion or addition */
   n_obj[sel] = n_obj[sel]-1;
/* option "3" is add a track segment which is assumed to be done */
if(option == 4) /* add object  ***/
   num_obj++;

/* In parameter matrix 1 for segment 0 for no segment,
   tracks are columns */
len_mask = 0;
for(r=0;r<num_obj;r++){
   if(iyproc <= n_obj[r] && ixproc == r)
   len_mask = 1;
}
dr = di = 0.0;
if(max_indx > -1){
   for(r=0;r<num_obj/2 + (num_obj-(num_obj/2)*2);r++){
      if(ixproc < n_obj[2*r]+1 ||
         (ixproc > 63 && ixproc < n_obj[2*r+1]+65)){
         cos_alpha = fp_cos(alpha[r]); /* speed up code by eliminating */
         cos_beta  = fp_cos(beta[r]);  /*   redundant calculations...   */
         sin_alpha = fp_sin(alpha[r]);
         sin_beta  = fp_sin(beta[r]);

         temp = pi*(index_d* cos_alpha*sin_beta +
                    index_u* cos_alpha*cos_beta);
         dr = dr + fp_cos(temp);
         di = di - fp_sin(temp);
      }
   }

   if(num_obj > 1 && ixproc < max_indx+1){
      dat_r_loc = dat_r - (S_R*xnetE[64].dr-S_I*xnetE[64].di);
      dat_i_loc = dat_i - (S_R*xnetE[64].di+S_I*xnetE[64].dr);
   }
   else if(ixproc < max_indx+1){
      dat_r_loc = dat_r;
      dat_i_loc = dat_i;
   }

   if(ixproc < max_indx+1){
      Sig_r = S_R*dr-S_I*di;
      Sig_i = S_R*di+S_I*dr;

      temp1 = (dat_r_loc - Sig_r);
      temp2 = (dat_i_loc - Sig_i);
      temp3 = -(temp1*temp1 + temp2*temp2)/(N_R*N_R+N_I*N_I);
      *likelihood = (fp_matsum(64,64,&temp3,128,0,0));
   }
```

```
      if(len_mask){
         range_temp1 = range_dat - fp_sqrt(X*X+Y*Y+Z*Z);
         range_temp2 = range_temp1*range_temp1;
         term = -fp_matsum(64,num_obj,&range_temp2,128,0,0);
      }
      *likelihood = *likelihood + term/(ran_var*ran_var);
   }
   else{
      *likelihood = -1000000.0; /* some large, negative number */
   }

   /* Reset the lengths to their original values  */
   if(option == 1)
      n_obj[sel] = n_obj[sel]+2;
   if(option == 2)
      n_obj[sel] = n_obj[sel]+1;
}

/***************************************************************************
   likelihood of track birth function
   determines the likelihood of adding a track (a new object) to the scene.
   This is a jump move.
***************************************************************************/
void
like_track_birth(int M, plural float dat_r,
                 plural float dat_i, plural float range_dat,
                 float *likelihood,float X0[OBJNUM_MAX],
                 float Y0[OBJNUM_MAX], float Z0[OBJNUM_MAX])
{
   plural float dr, di,Sig_r,Sig_i,temp,temp1,temp2,temp3;
   int r;
   float alpha[OBJNUM_MAX], beta[OBJNUM_MAX],
         pi, sin_alpha, sin_beta, cos_beta, cos_alpha;

   dr = 0; di = 0;
   pi = atan(1.0) * 4;

   for (r=0;r<M;r++) {
      beta[r] = (atan(Y0[r]/X0[r])) ;
      alpha[r] = (atan(Z0[r]/sqrt(X0[r]*X0[r] + Y0[r]*Y0[r]))) ;
      if(Y0[r] < 0){
         if(X0[r] < 0)
            beta[r] = beta[r] + pi;
         else
            beta[r] = beta[r] + 2*pi;
      }
      else{
         if(X0[r] < 0) beta[r] = beta[r] + pi;
      }
   }

   for (r=0;r<M;r++){
      if(ixproc < 1){
         cos_alpha = cos(alpha[r]);
         cos_beta  = cos(beta[r]);
         sin_alpha = sin(alpha[r]);
         sin_beta  = sin(beta[r]);

         temp = pi*(index_d* cos_alpha*sin_beta + index_u * cos_alpha*cos_beta);
         dr = dr + fp_cos(temp);
```

```
            di = di - fp_sin(temp);
         }
      }

      if(ixproc < 1){
         Sig_r = S_R*dr-S_I*di;
         Sig_i = S_R*di+S_I*dr;

         temp1 = (dat_r - Sig_r);
         temp2 = (dat_i - Sig_i);
         temp3 = -(temp1*temp1 + temp2*temp2)/(N_R*N_R+N_I*N_I);
         *likelihood = (fp_matsum(64,64,&temp3,128,0,0));
      }
   }
```

```
/***************************************************************************
   matrix_mult.m
***************************************************************************
   These comments last edited 6/27/95 by Jason August (jta1@cec.wustl.edu)

   These functions perform the indicated operations on matricies.
** End Introductory Comments */

/***************************************************************************
   Header Files to be included...
**************************************************************************/
#include <stdio.h>              /* standard i/o header */
#include <math.h>               /* math header */
#include <mpl.h>                /* MasPar programming language header */
#include <mpml.h>               /* MasPar mathematics library header */
#include <head.h>               /* the big header file that everything is in */
#include <param.h>              /* system parameter header file */
/* End Header Files to be included */

/***************************************************************************
 Global Variable Declarations
**************************************************************************/
extern int  num_obj;    /* number of estimated objects in the scene */

/***************************************************************************
   function: mymatmul
   T is multiplied by cov_out[obj_ind] and returned in K_x
**************************************************************************/
void
mymatmul(int n,
         int obj_ind,           /* current object */
         plural float T[3][3],
         plural float cov_out[OBJNUM_MAX][3][2],
         plural float K_x[OBJNUM_MAX][3][2])
{
   int             i, k, l, m;
   plural float    K_x_temp[3][3];
   register plural float temp, temp2[3];

   for (l = 0; l < 3; l++)
   {
      if (l == 0 && ixproc < 64)
      {
         temp2[0] = cov_out[obj_ind][0][0];
         temp2[1] = cov_out[obj_ind][1][0];
         temp2[2] = cov_out[obj_ind][2][0];
      } else
      if (l == 1 && ixproc < 64)
      {
         temp2[0] = xnetE[64].cov_out[obj_ind][0][0];
         temp2[1] = xnetE[64].cov_out[obj_ind][1][0];
         temp2[2] = xnetE[64].cov_out[obj_ind][2][0];
      } else
      if (l == 2 && ixproc < 64)
      {
         temp2[0] = cov_out[obj_ind][0][1];
         temp2[1] = cov_out[obj_ind][1][1];
         temp2[2] = cov_out[obj_ind][2][1];
      }
      for (k = 0; k < 3; k++)
      {
         temp = 0.0;
         for (i = 0; i < n + 1; i++)
         {
            if (iyproc == i && ixproc < 64)
            {
               temp = proc[i][obj_ind].T[k][0] * temp2[0] +
                  proc[i][obj_ind].T[k][1] * temp2[1] +
                  proc[i][obj_ind].T[k][2] * temp2[2];
               if (i > 0)
                  temp = temp + xnetN[1].temp;
            }
         }
         K_x_temp[k][l] = temp;
      }
   }

   for (k = 0; k < 3; k++)
   {
      for (l = 0; l < 3; l++)
      {
         temp = 0.0;
         for (i = 0; i < n + 1; i++)
         {
            if (ixproc == i)
            {
               temp = proc[i][obj_ind].T[k][0] * K_x_temp[l][0] +
                  proc[i][obj_ind].T[k][1] * K_x_temp[l][1] +
                  proc[i][obj_ind].T[k][2] * K_x_temp[l][2];
               if (i > 0)
                  temp = temp + xnetW[1].temp;
            }
         }
         if (l == 0 && ixproc < 64)
            K_x[obj_ind][k][0] = temp;
         if (l == 1 && ixproc > 63)
            K_x[obj_ind][k][0] = xnetW[64].temp;
         if (l == 2 && ixproc < 64)
            K_x[obj_ind][k][1] = temp;
      }
   }
}

/***************************************************************************
   function: mymatmul2
   T, a 3x3 matrix, is multiplied by cov_out, another 3x3 matrix,
   and returned in K_x
***************************************************************************/
void
mymatmul2(int n, plural float T[3][3], plural float cov_out[3][3],
          plural float K_x[3][3])
{
   int                i, k, l;     /* index variables */
   plural float       K_x_temp[3][3];
   register plural float temp;


   for (k = 0; k < 3; k++)
   {
      for (l = 0; l < 3; l++)
      {
         temp = 0.0;
         if (iyproc < n + 1)
            temp = T[k][0] * cov_out[0][1] + T[k][1] * cov_out[1][1] +
               T[k][2] * cov_out[2][1];
         for (i = 0; i < n + 1; i++)
         {
            if (iyproc == i)
            {
               if (i > 0)
                  temp = temp + xnetN[1].temp;
            }
         }
         K_x[k][1] = temp;
      }
   }


}

/**************************************************************************
   function: mymatvecmul
   multiplies a matrix K_x by a vector dX
**************************************************************************/
void
mymatvecmul(int n_obj[OBJNUM_MAX], plural float dX[3],
            plural float K_x[OBJNUM_MAX][3][2])
{
   int              i, k, r;
   register plural int address_loc;
   register plural float A, test, A2;
   plural float     dX_t[3];
   register plural int active;

   test = 0.0;

   /*** transpose the gradient vector and spread south ***/
   if (ixproc < 64)
      address_loc = nxproc * ixproc;
   else
      address_loc = 0;

   for (r = 0; r < num_obj; r++)
   {
      dX_t[0] = dX_t[1] = dX_t[2] = 0.0;
      if (iyproc == 0 && ixproc < n_obj[r] + 1)
      {
         dX_t[0] = router[address_loc + r].dX[0];
         dX_t[1] = router[address_loc + r].dX[1];
         dX_t[2] = router[address_loc + r].dX[2];
         xnetcS[63].dX_t[0] = dX_t[0];
         xnetcS[63].dX_t[1] = dX_t[1];
         xnetcS[63].dX_t[2] = dX_t[2];
      }
      /*** multiply with cov. mat. and sum along columns ***/
      for (i = 0; i < 3; i++)
      {
         A = A2 = 0.0;
         if (ixproc < n_obj[r] + 1)
         {
            A = K_x[r][i][0] * dX_t[0] + xnetE[64].K_x[r][i][0] * dX_t[1]
               + K_x[r][i][1] * dX_t[2];
         }
         /***  do matsumtovey ***/
         all              active = 0;
         active = 1;

         all
         {
            register int    d;
            if (!active)
               A = 0;            /* 0 = IDENT */
            if (ixproc < 64)
            {
               for (d = 1; d < 64; d = d << 1)
               {
                  if ((ixproc & d - 1) == d - 1)
                  {
                     A += xnetpW[d].A;
                  }
               }
            }
            if (ixproc == 63)
               xnetcW[63].A = A;
         }

         if (ixproc == r)
            dX[i] = A;

      }
   }
}
```

```
/***************************************************************************
   reshaping_covariance.m
****************************************************************************
   These comments last edited 6/26/95

   The following functions rearrange covariance matricies on the MasPar.
** End Introductory Comments */

/***************************************************************************
   Header Files to be included...
***************************************************************************/
#include <stdio.h>        /* standard i/o header */
#include <stdlib.h>       /* standard libraries... */
#include <math.h>         /* math header */
#include <mpl.h>          /* MasPar programming language header */
#include <mpml.h>         /* MasPar mathematics library header */

#include <head.h>         /* the big header file that everything is in */
#include <param.h>        /* system parameter header file */
/* End Header Files to be included */

/***************************************************************************
 Global Variable Declarations
***************************************************************************/
plural int      u_dep, v_dep, w_dep;

/***************************************************************************
   reshape_cov function
   to be computed efficiently by the MasPar, the covariance matrix cov_in
   must be shifted from datasets with clustered xyz coordinates to cov_out,
   a covariance matrix in which x, y, and z areas of data exist separately.
***************************************************************************/
void
reshape_cov(int n_obj[OBJNUM_MAX], int sel,
            plural float cov_in[OBJNUM_MAX][3][2],
            plural float cov_out[OBJNUM_MAX][3][2])
{
   int              i, k,
                    col, col2, element;
   plural int       addrT,
                    u_add, v_add, w_add;

   /* element = (3*n)%64; */
   col = (3 * n_obj[sel]) / 128;
   col2 = (3 * (n_obj[sel] + 1)) / 128;

   if (n_obj[sel] == 0)
   {
      /*** Setting the depths for u,v,w ****/
      if ((iyproc % 3) == 0)
         u_dep = 0;
      if ((iyproc % 3) == 1)
         u_dep = 2;
      if ((iyproc % 3) == 2)
         u_dep = 1;
      v_dep = xnetN[1].u_dep;
      if (iyproc == 0)
         v_dep = 1;
      w_dep = xnetS[1].u_dep;
      if (iyproc == 63)
      w_dep = 2;
}
/* address for router calls */
if (ixproc < 64)
   u_add = 128 * ((iyproc * 3) % 64) + 3 * ixproc % 128;
else
   u_add = xnetW[64].u_add + 1;
w_add = 128 * ((iyproc * 3) % 64) + (3 * ixproc + 2) % 128;


/* rearrange using router calls */
if (ixproc == n_obj[sel])
{
    /* u - u */
    cov_out[sel][0][0] = router[u_add].cov_in[sel][u_dep][col];

    /* v - u */
    cov_out[sel][1][0] = router[u_add + 128].cov_in[sel][v_dep][col];

    /* w - u */
    cov_out[sel][2][0] = router[u_add + 256].cov_in[sel][w_dep][col];
}
if (ixproc == n_obj[sel] + 64)
{
    /* u - v */
    cov_out[sel][0][0] = router[u_add].cov_in[sel][u_dep][col];

    /* v - v */
    cov_out[sel][1][0] = router[u_add + 128].cov_in[sel][v_dep][col];

    /* w - v */
    cov_out[sel][2][0] = router[u_add + 256].cov_in[sel][w_dep][col];

}
if (ixproc == n_obj[sel])
{
    /* u - w */
    cov_out[sel][0][1] = router[w_add].cov_in[sel][u_dep][col2];

    /* v - w */
    cov_out[sel][1][1] = router[w_add + 128].cov_in[sel][v_dep][col2];

    /* w - w */
    cov_out[sel][2][1] = router[w_add + 256].cov_in[sel][w_dep][col2];
}

/* transpose two columns into one row address
   for transposing the rearranged columns */
if (ixproc < 64)
   addrT = 128 * ixproc + n_obj[sel];
else
   addrT = xnetE[64].addrT;

if (iyproc == n_obj[sel])
{
   if (ixproc < 64)
   {
      cov_out[sel][0][0] = router[addrT].cov_out[sel][0][0];
      cov_out[sel][1][0] = router[addrT + 64].cov_out[sel][0][0];
      cov_out[sel][2][0] = router[addrT].cov_out[sel][0][1];
```

```
            cov_out[sel][0][1] = router[addrT].cov_out[sel][2][0];
            cov_out[sel][1][1] = router[addrT + 64].cov_out[sel][2][0];
            cov_out[sel][2][1] = router[addrT].cov_out[sel][2][1];
        } else
        {
            cov_out[sel][0][0] = router[addrT].cov_out[sel][1][0];
            cov_out[sel][1][0] = router[addrT + 64].cov_out[sel][1][0];
            cov_out[sel][2][0] = router[addrT].cov_out[sel][1][1];
        }
    }
}

/***************************************************************************
    shift_cov_out function
    shifts the covariances to cov_out
***************************************************************************/
void
shift_cov_out(int sel, plural float cov_out[OBJNUM_MAX][3][2])
{
    int i, k; /* indexing variables */

    /* shift the covariances */
    for (i = 0; i < 3; i++)
    {
        for (k = 0; k < 2; k++)
        {
            cov_out[sel][i][k] = xnetE[1].cov_out[sel][i][k];
            cov_out[sel][i][k] = xnetS[1].cov_out[sel][i][k];
        }
    }

}
```

```
/*************************************************************************
   shiftwindow.m
**************************************************************************
   These comments last edited 6/26/95 by Jason August (jta1@cec.wustl.edu)
** End Introductory Comments */

/*************************************************************************
   Header Files to be included...
*************************************************************************/
#include <stdio.h>      /* standard i/o header */
#include <math.h>       /* math header */
#include <mpl.h>        /* MasPar programming language header */
#include <mpml.h>       /* MasPar mathematics library header */

#include <head.h>       /* the big header file that everything is in */
#include <param.h>      /* system parameter header file */
/* End Header Files to be included */

/*************************************************************************
 Global Variable Declarations
*************************************************************************/
extern int      num_obj;

/*************************************************************************
   shiftwindow function:
   shifts the 64-length window to allow for newer data
*************************************************************************/
void
shiftwindow(int *shift_yes, int sel, int n_obj[OBJNUM_MAX],
            plural float *dat_r, plural float *dat_i,
            plural float *u, plural float *v, plural float *w,
            plural float *X, plural float *Y, plural float *Z,
            float X0[OBJNUM_MAX], float Y0[OBJNUM_MAX], float Z0[OBJNUM_MAX],
            plural float *theta, plural float *phi, plural float *psi,
            plural float *range_dat)
{
   int r; /* index variable */

   *shift_yes = 1; /* yes, shift */
   n_obj[sel] = 63;

   /* shift the 64-length window */
   *dat_r = xnetE[1].(*dat_r);
   *dat_i = xnetE[1].(*dat_i);
   *u = xnetS[1].(*u);
   *v = xnetS[1].(*v);
   *w = xnetS[1].(*w);

   for (r = 0; r < num_obj; r++)
   {
      X0[r] = proc[0][r].(*X);
      Y0[r] = proc[0][r].(*Y);
      Z0[r] = proc[0][r].(*Z);
   }

   *theta = xnetS[1].(*theta);
   *phi = xnetS[1].(*phi);
   *psi = xnetS[1].(*psi);
   *range_dat = xnetS[1].(*range_dat);
   if (ixproc == 63)
      *dat_r = *dat_i = 0.0;

   if (iyproc == 63)
   {
      *u = *v = *w = 0.0;
      *X = *Y = *Z = 0.0;
      *theta = *phi = *psi = 0.0;
      *range_dat = 0.0;
   }

   /* decrement the n_obj values for other tracks */
   for (r = 0; r < num_obj; r++)
   {
      n_obj[r] = n_obj[r] - 1;
   }

   n_obj[sel] = n_obj[sel] + 1;
}
```

```
/****************************************************************************
   write_socket.m
*****************************************************************************
   These comments last edited 6/26/95 by Jason August (jta1@cec.wustl.edu)
** End Introductory Comments */

/****************************************************************************
   Header Files to be included...
****************************************************************************/
#include <stdio.h>      /* standard i/o header */
#include <math.h>       /* math header */
#include <mpl.h>        /* MasPar programming language header */
#include <mpml.h>       /* MasPar mathematics library header */
#include <su/unistd.h>
#include <head.h>       /* the big header file that everything is in */
#include <param.h>      /* system parameter header file */
/* End Header Files to be included */

/****************************************************************************
 Global Variable Declarations
****************************************************************************/
extern float    track[OBJNUM_MAX][tracksize][3],
                pitch[OBJNUM_MAX][tracksize],
                yaw[OBJNUM_MAX][tracksize],
                roll[OBJNUM_MAX][tracksize];

extern int      num_obj;

/****************************************************************************
   write_socket function
   moves recently computed data to display SGI
****************************************************************************/
void
write_socket(int num_obj, int sock1,
             int len_obj[OBJNUM_MAX], int n_obj[OBJNUM_MAX],
             int len_track, int move_type, int accept_reject,
             plural float X, plural float Y, plural float Z,
             int index[OBJNUM_MAX]) {
   int i, r; /* index variables */
   float socket_temp[192]; /* socket data buffer */

   /* write to the sockets */
   write(sock1, &num_obj, sizeof(num_obj));
   write(sock1, &len_track, sizeof(len_track));
   write(sock1, &move_type, sizeof(move_type));
   write(sock1, &accept_reject, sizeof(accept_reject));

   for (r = 0; r < num_obj; r++)
   {
      write(sock1, &len_obj[index[r]], sizeof(int));
      write(sock1, &n_obj[index[r]], sizeof(int));
   }

   for (r = 0; r < num_obj; r++)
   {
      for (i = 0; i <= 63; i++)
      {
         socket_temp[3 * i] = proc[i][index[r]].Y;
         socket_temp[3 * i + 1] = proc[i][index[r]].Z;
         socket_temp[3 * i + 2] = proc[i][index[r]].X;
```

```
        }
        write(sock1, socket_temp, 192 * sizeof(float));
    }
}
```

The current directory is:

```
~atr/demo/display
```

This directory contains the C program

```
big.c
```

Which is responsible for displaying the results of target detection and tracking using simulated data from a cross array of radar sensors. The program receives true and estimated track information from the mpp program via a socket connection, and uses routines from the standard SGI graphics library to produce a 3-D rendering of the simulated active scene. See the source code for further comments.

```
/****************************************************************
       big.c

This is a display program which displays the true and the estimated
configurations using the SGI GL. It receives all the parameters
(true & estimated) across the socket from MPP programs.
****************************************************************/


/****************************************************************
                header files
****************************************************************/
#include <stdio.h>
#include <math.h>
#include <gl/gl.h>
#include <gl/device.h>
#include <fcntl.h>
#include <strings.h>
#include <fmclient.h>
#include <OBJ.h>
#include <flip.h>
#include <realparms.h>

/****************************************************************
Some fixed parametes which should be defined in the header file
****************************************************************/
#define X 0
#define Y 1
#define Z 2
/* #define vert_down_shift -2000 */
#define vert_down_shift 0
#define size2 40
#define OTHOX 3
#define OTHOY 3
#define OTHOZ 5


/****************************************************************
        global variables for the display program
****************************************************************/

/*** True parameters read from a file in the mpp program and
     sent over a socket for display of true configuration ****/

    float yaw[OBJNUM_TRU][tracksize],
        pitch[OBJNUM_TRU][tracksize],
        roll[OBJNUM_TRU][tracksize],
        track_read[OBJNUM_TRU*tracksize][3],
        track[OBJNUM_TRU][tracksize][3],
        track1[OBJNUM_TRU][tracksize][3],

/****** Estimated positions forming the tracks ***/

        track1_est[OBJNUM_TRU][tracksize][3],
        track_est[OBJNUM_TRU][tracksize][3];

/*** Window ids ***/

    long mainwin,
        smallwin[OBJNUM_TRU],
          smallerwin[OBJNUM_TRU];

      float x_0[OBJNUM_TRU],
          y_0[OBJNUM_TRU],
          z_0[OBJNUM_TRU];

  /* Window coordinates used by SGI for display */

      long xorigin,
          yorigin;

  /* Current estimated length of the tracks */

      int current_len[OBJNUM_TRU];

  /* Actual length of the tracks */

      int len_track = 0;

  /* Total estimated number of targets in the scene */

      int num_obj = 0;

  /** data type for x29 display */

      flip_type flips[1];

  /*** Viewing matrix for display, initialized to identity **/

      Matrix Identity = { 1, 0, 0, 0,  0, 1, 0, 0,
          0, 0, 1, 0,  0, 0, 0, 1 };

  /******* Definition of display, lighting, materials etc for graphics **/

      static float dullmat[] = {
          AMBIENT, 0.1, 0.1, 0.2,
          DIFFUSE, 0.2, 0.4, 0.8,
          SPECULAR, 0.2, 0.2, 0.3,
          SHININESS, 30,
          LMNULL
      };

      static float shufmat[] = {
          AMBIENT, 0.1, 0.2, 0.1,
          DIFFUSE, 0.5, 0.9, 0.4,
          SPECULAR, 0.2, 0.4, 0.2,
          SHININESS, 65,
          LMNULL
      };

      static float mat[] = {
          AMBIENT, 0., 0., 0.,
          DIFFUSE, 0.5, 0.5, 0.5,
          SPECULAR, 1, 1, 1,
          SHININESS, 64,
          LMNULL,
      };

      static float lm[] = {
          AMBIENT, .1, .1, .1,
        LOCALVIEWER, 1,
        LMNULL
    };

    static float lt_front[] = {
        LCOLOR, 1., 1., 1.,
        POSITION, 0., 0., 1., 0.,
        LMNULL
    };

/***********************************************************************
        The main program
***********************************************************************/
main()
{
    long xsize,ysize;   /* temp space for window size              */
    float xrot,yrot,B_yrot=0;
    short val;          /* temp space for monitoring input         */
    int dev;            /* temp space for input device             */
    int  sock,flag;
    int i,j,k,n,l,num[OBJNUM_TRU],r;
    float a;
    char name[80];
    FILE *fp;
    fmfonthandle font1, font25, font15;
    char fname0[80];
    int max_current_len, move_type, accept_reject;


/* setup font stuff for text writitng in the graphic windows */

    fminit();
    if ((font1=fmfindfont("Times-Roman")) == 0)
        exit(1);
    font25 = fmscalefont(font1,25.0);
    font15 = fmscalefont(font1,15.0);
    fmsetfont(font25);

/* Read in the X29 surface description file for X29 display*/

    sprintf(fname0,"/usr/people/spj/radar/plane.bin");
    flip_read(fname0,&flips[0]);
    printf("Read the data files \n");

/* open windows and setup graphics environment */

/**** This window shows the global scene simulation ***/

    noborder();
    prefposition(0,600,500,750);
    mainwin = winopen("plane_track");
    lmdef(DEFLIGHT, 1, 0, lt_front);
    lmdef(DEFLMODEL, 1, 0, NULL);
    lmdef(DEFMATERIAL, 1, 0, dullmat);
    lmbind(LMODEL, 1);
    lmbind(LIGHT0, 1);
    lmbind(MATERIAL, 1);
    qdevice(ESCKEY);    /* monitor for ESC key to exit pgm         */
    RGBmode();
    doublebuffer();
    gconfig();
    lsetdepth(0, 0x7fffff);
    zbuffer(TRUE);
    mmode(MVIEWING);
    loadmatrix(Identity);
    getorigin(&xorigin,&yorigin);
    getsize(&xsize,&ysize);
    ortho(-13000,13000,-4000,7000,-13000,13000);
    czclear(0x000000, 0x7fffff);
    swapbuffers();


    for(r=0;r<OBJNUM_TRU; r++){

/****  Smaller windows for showing magnified views of the tracks
       to emphasize the tracking process ******/

        noborder();
        prefposition(400,600,300-r*200,500-r*200);
        smallwin[r] = winopen("magnified_track");
        winset(smallwin[r]);
        lmdef(DEFLIGHT, 1, 0, lt_front);
        lmdef(DEFLMODEL, 1, 0, NULL);
        lmdef(DEFMATERIAL, 1, 0, dullmat);
        lmbind(LMODEL, 1);
        lmbind(LIGHT0, 1);
        lmbind(MATERIAL, 1);
        qdevice(ESCKEY);
        RGBmode();
        doublebuffer();
        gconfig();
        lsetdepth(0, 0x7fffff);
        zbuffer(TRUE);
        mmode(MVIEWING);
        loadmatrix(Identity);
        getorigin(&xorigin,&yorigin);
        ortho(-OTHOX,OTHOX,-OTHOY,OTHOY,-OTHOZ,OTHOZ);
        czclear(0x000000, 0x7fffff);
        swapbuffers();

    }

/* Initialize the socket for receiving true tracks and estimates
   from the maspar program for simultaneous display of results */

    sock = recvinit_p(1995);
    if(sock == -1)
        printf("Big: Error Socket not established\n");
    printf("Big: Socket received ..!!\n"); fflush(stdout);

/*** Receives the complete true tracks for the display purpose ***/

    while (recvmine(sock,&flag,sizeof(int)) != 1);
    swap_it(&flag);
    if(flag == 1)
    {
        for(r = 0; r < OBJNUM_TRU; r++)
        {
            for(i=0;i< tracksize; i++)
            {
                read_sock(sock,&track_read[3*r*tracksize+i][X],
                    sizeof(float));
                read_sock(sock,&track_read[3*r*tracksize+i][Y],
                    sizeof(float));
                read_sock(sock,&track_read[3*r*tracksize+i][Z],
                    sizeof(float));
                swap_it(&track_read[3*r*tracksize+i][X]);
                swap_it(&track_read[3*r*tracksize+i][Y]);
                swap_it(&track_read[3*r*tracksize+i][Z]);
            }
        }

        read_sock(sock,&pitch[0][0],OBJNUM_TRU*tracksize*sizeof(float));
        read_sock(sock,&yaw[0][0],OBJNUM_TRU*tracksize*sizeof(float));
        read_sock(sock,&roll[0][0],OBJNUM_TRU*tracksize*sizeof(float));

        for(r = 0; r < OBJNUM_TRU; r++)
        {
            for(i=0;i< tracksize; i++)
            {
                swap_it(&pitch[r][i]);
                swap_it(&yaw[r][i]);
                swap_it(&roll[r][i]);
            }
        }

    }

    printf("Big: Actual Track read and formatted\n");

/*** Restructuring the data to match the display routines ***/

    for(r= 0; r< OBJNUM_TRU; r++)
    {
        for(i=0;i<tracksize-1;i++)
        {
            track[r][i][X] = (track_read[3*r*tracksize+i][1])/100;
            track[r][i][Y] = (track_read[3*r*tracksize+i][2])/100;
            track[r][i][Z] = (track_read[3*r*tracksize+i][0])/100;
            track1[r][i][X] = (track_read[3*r*tracksize+i][1]);
            track1[r][i][Y] = 2.5*(track_read[3*r*tracksize+i][2]);
            track1[r][i][Z] = (track_read[3*r*tracksize+i][0]);
        }

        fclose(fp);
        printf("Big: Reading done\n"); fflush(stdout);

 /***** Initial track segment *****/

        track1_est[r][0][X] = track1[r][0][X];
        track1_est[r][0][Y] = track1[r][0][Y];
        track1_est[r][0][Z] = track1[r][0][Z];
        track_est[r][0][X] = track[r][0][X];
        track_est[r][0][Y] = track[r][0][Y];
        track_est[r][0][Z] = track[r][0][Z];
    }

/* The while loop which executes during the estimate display */

    len_track = tracksize;
    while (!(qtest() && (dev=qread(&val)) == ESCKEY && val == 0))
    {
        if(max_current_len == 1000 )
            exit(1);

/**** Receiving the estimates of various state variables across
      the socket from the mpp estimator *******/

        if(max_current_len < tracksize-2)
        {
            while (recvmine(sock,&num_obj,sizeof(num_obj)) != 1);
            swap_it(&num_obj);
            read_sock(sock,&len_track,4);
            swap_it(&len_track);
            read_sock(sock,&move_type,4);
            swap_it(&move_type);
            read_sock(sock,&accept_reject,4);
            swap_it(&accept_reject);

            for(r=0;r<num_obj; r++)
            {
                read_sock(sock,&current_len[r], 4);
                swap_it(&current_len[r]);
                read_sock(sock, &num[r], 4);
                swap_it(&num[r]);
            }

            for(r=0;r<num_obj; r++)
            {
                for(k=0;k<=63;k++)
                {
                    for(i=0;i<3;i++)
                    {
                        read_sock(sock,&a,sizeof(float));
                        swap_it(&a);
                        if(i == 0)
                        {
                            track1_est[r][current_len[r]-num[r]+k][X]
                                = (a - 0*x_0[r]);
                            track_est[r][current_len[r]-num[r]+k][X]
                                = (a-x_0[r])/100;
                        }
                        else if(i == 1)
                        {
                            track1_est[r][current_len[r]-num[r]+k][Y]
                                = (a - 0*y_0[r])* 2.5;
                            track_est[r][current_len[r]-num[r]+k][Y]
                                = (a-y_0[r])/100;
                        }
                        else
                        {
                            track1_est[r][current_len[r]-num[r]+k][Z]
                                = (a - 0*z_0[r]);
                            track_est[r][current_len[r]-num[r]+k][Z]
                                = (a-z_0[r])/100;
                        }
                    }
                }
            }
        }
        max_current_len = max(current_len[0],current_len[1]);

        if (dev == REDRAW)
        {
            for(r=0;r<OBJNUM_TRU; r++)
            {
                winset(smallwin[r]);
                winset(smallerwin[r]);
                reshapeviewport();
            }
            winset(mainwin);
            reshapeviewport();
        }

/* draw in main window */

        winset(mainwin);
        dev = 0;
        qreset();
        czclear(0x000000, 0x7fffff);
        pushmatrix();

/***** Text in the window *****/

        RGBcolor(0,255,255);
        cmovi(-8000,5500,0);
        fmprstr("MULTI-TARGET TRACKING");

        fmsetfont(font15);
        RGBcolor(255,255,0);
        cmovi(5700,-2500,0);

        if(move_type == 3)
            fmprstr("TRACK BIRTH");
        else if(move_type == 1)
            fmprstr("SEGMENT BIRTH");
        else if(move_type == 2)
            fmprstr("SEGMENT DEATH");

        if(accept_reject == 1)
        {
            RGBcolor(0,255,0);
            cmovi(5700,-3500,0);
            fmprstr("ACCEPTED");
        }
        else if(accept_reject == 2)
        {
            RGBcolor(255,0,0);
            cmovi(5700,-3500,0);
            fmprstr("REJECTED");
        }

        RGBcolor(128,0,255);
        cmovi(-12000,-3000,0);

        if(num_obj == 0)
            fmprstr("TARGETS = 0");
        if(num_obj == 1)
            fmprstr("TARGETS = 1");
        if(num_obj == 2)
            fmprstr("TARGETS = 2");

        fmsetfont(font25);
        xrot = 15;
        B_yrot  = B_yrot + 1;;
        rot(xrot,'x');
        rot(B_yrot,'y');
        drawtrack();
        drawgrid();
        axes();
        popmatrix();
        whiteline1();
        swapbuffers();

/* draw in small window */

        for(r=0;r< OBJNUM_TRU; r++)
        {
            winset(smallwin[r]);
            czclear(0x000000, 0x7fffff);
            pushmatrix();

/***** Text in the window *****/

            fmsetfont(font15);
            RGBcolor(0,255,255);
            cmov(-1.2,2.0,0);
            sprintf(name,"TRACK %d",r+1);
            fmprstr(name);
            xrot = 15;
            yrot = 135;
            rot(xrot,'x');
            rot(yrot,'y');
            axes2();
            drawsmallwin(r);
            popmatrix();
            if(r == OBJNUM_TRU -2 )
                whiteline2();
            swapbuffers();
        }

/* End of loop to draw magnified tracks in snall windows */

    }

/*********** End of the while loop for estimate display ****/

}

/**  End of the main program *****/


/*********************************************************************/
/********** The subroutine for drawing the inertial axes system in the
                main window *****/
axes()
{
    float x_axis[3],y_axis[3],z_axis[3];
    float origin[3];
    int i,j,k;
    Object axes;


/* naming the axes       */
    cpack(0xFFFFFFFF);
    cmov(-4000,0*200 + vert_down_shift,0);
    charstr("X");
    cmov(0,4000 + vert_down_shift,0);
    charstr("Y");
    cmov(0,0*200 + vert_down_shift,-4000);
    charstr("Z");

/* Draw the principal axes */
    linewidth(1);
    cpack(0xFF8F00FF);
    origin[X] = origin[Z] = 0.0;
    origin[Y] = 0*200 + vert_down_shift;
    x_axis[X] = z_axis[Z] = -4000.0;
    y_axis[Y] = 4000 + vert_down_shift;
    x_axis[Z] = z_axis[X] = 0.0;
    y_axis[X] = y_axis[Z] = 0.0;
    x_axis[Y] = 0*200.0 + vert_down_shift;
    z_axis[Y] = 0*200.0 + vert_down_shift;

    linewidth(3);
    makeobj(axes);
    bgnline();
    v3f(origin);
    v3f(x_axis);
    endline();
    bgnline();
    v3f(origin);
    v3f(y_axis);
    endline();
    bgnline();
    v3f(origin);
    v3f(z_axis);
    endline();
    closeobj();
    callobj(axes);

}
/*************************************************************************/
drawgrid()
{
    int i;
    float point[3];

/* draw grid at ground plane */

    linewidth(1);
    cpack(0xFF0F8800);
    point[Y] = vert_down_shift;
    for (i=-BSIZE; i<=BSIZE; i+=500)
    {
        bgnline();
            point[X] = i;
            point[Z] = -BSIZE;
            v3f(point);
            point[Z] = -point[Z];
            v3f(point);
        endline();

        bgnline();
            point[Z] = i;
            point[X] = -BSIZE;
            v3f(point);
            point[X] = -point[X];
            v3f(point);
        endline();
    }

/* draw radar array */

    cpack(0xFFFFFFFF);
    linewidth(5);
    for (i=-2000; i<=2000; i+=500)
    {
        bgnline();
            point[Z] = i-75;
            point[X] = 0;
            v3f(point);
            point[Z] = i+75;
            v3f(point);
        endline();

        bgnline();
            point[X] = i-75;
            point[Z] = 0;
            v3f(point);
            point[X] = i+75;
            v3f(point);
        endline();
    }

}

/***********************************************************************/
drawtrack()
{
    int i,r;
    float local[tracksize][3];
    float local_est[tracksize][3];

    for(r=0;r<OBJNUM_TRU; r++)
    {

/* true track */

        linewidth(3);

        if(r == 0)
            cpack(0xFF0000FF);
        else
            cpack(0xFFFF0000);

        bgnline();
            for (i=1; i<tracksize-1; i++)
            {
                local[i][X] = track1[r][i][X];
                local[i][Y] = track1[r][i][Y];
                local[i][Z] = track1[r][i][Z];
                v3f(local[i]);
            }
        endline();

/* estimated track*/

        linewidth(2);
        cpack(0xFF00FFFF);
        zfunction(ZF_ALWAYS);

        bgnline();
            for (i=1; i<=current_len[r]; i++)
            {
                local_est[i][X] = track1_est[r][i][X];
                local_est[i][Y] = track1_est[r][i][Y];
                local_est[i][Z] = track1_est[r][i][Z];
                v3f(local_est[i]);
            }
        endline();

        zfunction(ZF_LEQUAL);

    }
}

/*******************************************************************/
drawsmallwin(int r)
{
    int i,k,n,l,j;
    float local[size2+size2/2][3];
    float local_est[size2][3];
    float scl=0.6;
    int local_int,meanx,meany,meanz;

    j = current_len[r];
    k = j/size2;

    if(j < (len_track/size2)*size2 || (j < len_track-1))
        local_int = size2;
    else
        local_int = len_track - (len_track/size2)*size2;

/*****************TRUE ***************/

    linewidth(5);
    if(r == 0)
        cpack(0xFF0000FF);
    else
        cpack(0xFFFF0000);

    bgnline();
        if(local_int-1 < size2)
        {
            meanx = track[r][k*size2+local_int/2-1][X];
            meany = track[r][k*size2+local_int/2-1][Y];
            meanz = track[r][k*size2+local_int/2-1][Z];
        }
        else
        {
            meanx = track[r][k*size2+size2/2][X] ;
            meany = track[r][k*size2+size2/2][Y] ;
            meanz = track[r][k*size2+size2/2][Z] ;
        }

        for(i=1;i<local_int;i++)
        {
            local[i][X] = track[r][i+k*size2][X] - meanx;
            local[i][Y] = track[r][i+k*size2][Y] - meany;
            local[i][Z] = track[r][i+k*size2][Z] - meanz;
            v3f(local[i]);
        }
    endline();

/************ ESTIMATE *****************/

    linewidth(3);
    cpack(0xFF00FFFF);
    zfunction(ZF_ALWAYS);
    bgnline();
        l = j+1;
        for(i=0;i<(l-k*size2);i++)
        {
            local_est[i][X] = track_est[r][i+k*size2][X] - meanx;
            local_est[i][Y] = track_est[r][i+k*size2][Y] - meany;
            local_est[i][Z] = track_est[r][i+k*size2][Z] - meanz;
            v3f(local_est[i]);
        }
    endline();
    zfunction(ZF_LEQUAL);

}

axes2()
{
    float x_axis[3],y_axis[3],z_axis[3];
    float origin[3];
    int i,j,k;
    Object axes;

/* naming the axes      */

    cpack(0xFFFFFFFF);
    cmov(-1.0,0*200 -0.0,0);
    charstr("X");
    cmov(0,1.0  -0.0,0);
    charstr("Y");
    cmov(0,0*200  -0.0 ,-1.0);
    charstr("Z");

/* Draw the principal axes */

    linewidth(1);
    cpack(0xFF8F00FF);
    origin[X] = origin[Z] = 0.0;
    origin[Y] = 0*200 -0.0;
    x_axis[X] = z_axis[Z] = -1.0;
    y_axis[Y] = 1.0 -0.0;
    x_axis[Z] = z_axis[X] = 0.0;
    y_axis[X] = y_axis[Z] = 0.0;
    x_axis[Y] = 0*200.0  -0.0;
    z_axis[Y] = 0*200.0  -0.0;

    linewidth(3);
    makeobj(axes);
        bgnline();
            v3f(origin);
            v3f(x_axis);
        endline();
        bgnline();
            v3f(origin);
            v3f(y_axis);
        endline();
        bgnline();
            v3f(origin);
            v3f(z_axis);
        endline();
    closeobj();
    callobj(axes);

}



/***********************************************************************/
int swap_it(data)
unsigned char *data;
{
    unsigned char cptr[2];

    cptr[0] = data[0];
    cptr[1] = data[1];

    data[0] = data[3];
    data[1] = data[2];
    data[2] = cptr[1];
    data[3] = cptr[0];

}

/*****************************************************************/
int read_sock(sock,ptr,sz)
int sock,sz;
char *ptr;
{
    int rd=0;
    int ret;

    while (rd < sz) {
        ret = read(sock,ptr,sz-rd);
        if (ret <= 0) return(ret);
        ptr += ret;
        rd += ret;
    }
    return(rd);
}

/***********************************************************************/
```

```
int whiteline2()
{
    int i;
    float vert[2][3];

    linewidth(3);
    cpack(0xFFFFFFFF);
    bgnline();
        for(i=0;i<2;i++)
        {
            vert[i][X] = 0.99*OTHOX;
            vert[i][Y] = (2*i-1)*OTHOY;
            vert[i][Z] = 0;
            v3f(vert[i]);
        }
    endline();
}

int whiteline1()
{
    int i;
    float vert[2][3];

    linewidth(2);
    cpack(0xFFFFFFFF);
    bgnline();
        for(i=0;i<2;i++)
        {
            vert[i][X] = (2*i-1)*13000;
            vert[i][Y] = -3900;
            vert[i][Z] = 0;
            v3f(vert[i]);
        }
    endline();
}
```

The current directory is

~atr/demo/faisalsrc

This directory contains routines for performing orientation estimation using simulated range profile data, and for sending the results over a socket connetction to a display program. The estimation is performed by performing a local search over a grid of pitch and yaw angles, using a dictionary of range profiles which have been simulated for each orientation in the grid. The following routines are contained in this directory:

PY_to_PE.m	converts orientation to dictionary address
arc_angle.m	computes the angle subtended between two points on the unit sphere
f_init_jupiter.m	loads range profile dictionary and establishes local socket connection
f_initialize.m	loads range profile dictionary and establishes long-distance socket connection
fix_pitch_angle.m	ensures -180 <= pitch < 180
get_pitch.m	2 angle -> 3 angle orientation conversion returns pitch angle
get_pitch_prime.m	3 angle -> 2 angle orientation conversion returns pitch angle
get_yaw.m	2 angle -> 3 angle orientation conversion returns yaw angle
get_yaw_prime.m	3 angle -> 2 angle orientation conversion returns yaw angle
load_rp.m	loads individual range profiles
min2.m	minimum of 2 floats
nearest_rp.m	point on sphere -> nearest grid point
polar_to_cart.m	polar coordinates -> cartesian
round.m	rounds float to int
rp_diff.m	summed square difference between range profiles
search.m	performs local search over orientation grid calls other routines including trace
simulate.m	extracts observed range profile from library
slope.m	computes gradient of likelihood surface between two grid points
sq_chord.m	computes chord length between two points on the unit sphere
trace.m	performs local search over orientation grid

See the source code for further comments.

```
/*
   Author: Mohammad Faisal

   Given 2 angle orientation of the target plane, this routine locates
   the PE and the memory depth where the corresponding dictionary range
   profile is located.  The mapping from the 2 angles to the PE number and
   memory depth is one-to-one.
*/

#include <stdio.h>
#include <math.h>
#include "faisalheader.h"

void PY_to_PE(f_pitch,f_yaw,sum_pitch_vect,num_yaw,delta_yaw,pitch_vect, \
              delta_pitch_vect,rp_per_layer,proc_num,mem,bins)

  float *f_pitch,*f_yaw,*delta_yaw,*delta_pitch_vect;
  int *num_yaw,*sum_pitch_vect,*pitch_vect,*rp_per_layer,*proc_num, \
      *bins,*mem;

{
  int rp_num,yaw_index;

  if (round(*f_yaw / *delta_yaw) < 0) {
    yaw_index = round(fabs(*f_yaw) / *delta_yaw); /* start 0 */
    rp_num = *(sum_pitch_vect + *num_yaw - 1); /* start 1 */
    if (yaw_index > 1)
      rp_num += *(sum_pitch_vect+yaw_index-1) - *pitch_vect;
  }
  else {
    yaw_index = round(*f_yaw / *delta_yaw);
    if (yaw_index > 0)
      rp_num = *(sum_pitch_vect+yaw_index-1);
    else
      rp_num = 0;
  }

  if (round(*f_pitch / *(delta_pitch_vect+yaw_index)) < 0) {
    rp_num += (*(pitch_vect+yaw_index) + 1)/2;
    rp_num += round(fabs(*f_pitch) / *(delta_pitch_vect+yaw_index));
  }
  else
    rp_num += round(*f_pitch / *(delta_pitch_vect+yaw_index))+1;

  *mem = (rp_num-1) / *rp_per_layer; /* start 0 */
  *proc_num = ((rp_num-1) % *rp_per_layer) * *bins; /* start 0 */
}
```

```
/*
   Author: Mohammad Faisal

   Given a pair of points on the sphere, this routine computes the arc angle
   between the two points assuming a unit radius sphere.

   In this routine variable names containing pitch implies the elevation of the
   sphere and yaw implies the azimuth of the sphere which should not be
   confused with the pitch and yaw angles of the target plane.
*/

#include <stdio.h>
#include "math.h"
#include "grad_search.h"
#include "faisalheader.h"


float arc_angle(pitch1,yaw1,pitch2,yaw2)

  float *pitch1,*yaw1,*pitch2,*yaw2;

{
  return(2.0 * asin(sqrt(sq_chord(pitch1,yaw1,pitch2,yaw2))/2.0) \
         * 360.0 / (2.0 * PI));
}
```

```
/*
   Author: Mohammad Faisal

   This is the initialization routine for estimating the orientation of the
   target plane given a range profile representing the data for the
   orientation to be estimated and a given a dictionary of range profiles
   of the target plane. The target plane used is the X29.

   This subroutine can load the individual simulated noiseless range
   profiles generated by the software called XPATCH, and once loaded,
   can save them as a binary file for fast loading.  Whether to lead the
   range profiles individually from XPATCH output files or from pre-stored
   binary file is determined by the variable RP_IN_DICT.

   It opens a socket connection to an SGI machine for displaying the
   estimated orientations. Several other variables are initialized and several
   data structures are computed to be used during the estimation.

   All angles are in degrees.
   Number of range bins in a range profile must be equal or laess than nproc.
   -90 <= yaw <= 90
   -180 <= pitch < 180
   -180 <= roll <= 180
*/

#include <stdio.h>
#include <math.h>
#include <mpl.h>
#include <mpipl.h>
#include "grad_search.h"
#include <stdlib.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/file.h>
#include <sys/types.h>
#include <ppeio.h>
#include "faisalheader.h"

void f_init_jupiter()
{
  extern int close(int);
  visible extern int sendinit_jupiter();
  extern int num_yaw,count,bins,total_num_rp,rp_per_layer,layers, \
      p_count,y_count,mem,proc_num,are_rps_a_file,want_to_save_dict,\
      rp_count,fd;
  extern int *pitch_vect,*sum_pitch_vect,k;
  extern char dir_name[200],data_filename[200],rp_dict_filename[200], \
       dummy_string[200];
  extern float init_pitch,init_yaw,delta_yaw,f_pitch,f_yaw,dummy,val, \
        found_pitch_prime,found_yaw_prime,init_roll,init_pitch_prime, \
        init_yaw_prime,found_pitch,found_yaw;
  extern float *delta_pitch_vect;
  extern float true_pitch,true_yaw,true_roll,true_pitch_prime,true_yaw_prime;
  extern int true_proc_num,true_mem;
  extern plural float *rp_dictionary,*f_data;

  /* Variable for socket comm. */
  extern int f_sock;

  /* Open socket */
```

```
f_sock = callRequest(sendinit_jupiter,4,1989);
        if (f_sock == -1) {
        printf("socket error THESEUS socket error\n");
        exit(1); }
        else printf("THESEUS: socket has been opened\n");

num_yaw = DISCRETIZE_PAR1;

/* Dynamically allocate memory */
pitch_vect = (int *) malloc(num_yaw*sizeof(int));
sum_pitch_vect = (int *) malloc(num_yaw*sizeof(int));
delta_pitch_vect = (float *) malloc(num_yaw*sizeof(int));

printf("Done allocating memory\n");

delta_yaw = 90.0/(num_yaw-1);

k = DISCRETIZE_PAR2;

for (count=0;count<num_yaw;++count) {
  if (count < (num_yaw-1))
    *(pitch_vect+count) = round(k * cos(delta_yaw * count * 2.0 * PI \
                                        / 360.0));
  else
    *(pitch_vect+count) = 1;
  if (count==0) {
    *sum_pitch_vect = *pitch_vect;
  }
  else {
    *(sum_pitch_vect+count) = *(sum_pitch_vect+count-1) + \
                              *(pitch_vect+count);
  }
  *(delta_pitch_vect+count) = 360.0 / *(pitch_vect+count);
}
total_num_rp = (2 * *(sum_pitch_vect+num_yaw-1)) - *pitch_vect;
bins = NUM_BINS;
are_rps_a_file = RP_IN_DICT;

rp_per_layer = nproc / bins;
layers = ceil(((float) total_num_rp) / ((float) rp_per_layer));

/* Dynamically allocate memory to store range profile dictionary */
rp_dictionary = (plural float *) p_malloc(layers*sizeof(float));

printf("Ready to load dictionary\n");
/* loading dictionary in PEs */
if (are_rps_a_file) {

  /* Range profile dictionary already exists as one file on disk.
     Load it in memory */
  sprintf(rp_dict_filename, \
  "/recognition/people/atr/demo/faisalsrc/data/x29_data/x29_dict.float");
  fd = open(rp_dict_filename,O_RDONLY,0);
  if (iproc < (rp_per_layer * bins))
    p_read(fd,rp_dictionary,layers * 4);
  close(fd);
}
else {

  /* Range profile dictionary does not exist as one file on disk.
```

```
   Load them one by one using the range profile file naming convention */
printf("Enter directory name for dictionary? ");
scanf("%s\n",dir_name);

count = 0;
f_yaw = 0.0;
for (y_count=0;y_count<num_yaw;++y_count) {

  f_pitch = 0.0;
  for (p_count=0;p_count<((*(pitch_vect+y_count)+1)/2);++p_count) {

    PY_to_PE(&f_pitch,&f_yaw,sum_pitch_vect,&num_yaw,&delta_yaw, \
             pitch_vect,delta_pitch_vect,&rp_per_layer, \
             &proc_num,&mem,&bins);
    load_rp(&f_pitch,&f_yaw,&mem,&proc_num,dir_name,&bins,rp_dictionary);

    if (y_count > 0) {
      f_yaw = -f_yaw;
      PY_to_PE(&f_pitch,&f_yaw,sum_pitch_vect,&num_yaw,&delta_yaw, \
               pitch_vect,delta_pitch_vect,&rp_per_layer, \
               &proc_num,&mem,&bins);
      load_rp(&f_pitch,&f_yaw,&mem,&proc_num,dir_name,&bins,rp_dictionary);
      f_yaw = -f_yaw;
    }

    f_pitch += *(delta_pitch_vect+y_count);
  }

  f_pitch = -(*(delta_pitch_vect+y_count));
  for (p_count=0;p_count<(*(pitch_vect+y_count)/2);++p_count) {

    PY_to_PE(&f_pitch,&f_yaw,sum_pitch_vect,&num_yaw,&delta_yaw, \
             pitch_vect,delta_pitch_vect,&rp_per_layer, \
             &proc_num,&mem,&bins);

    load_rp(&f_pitch,&f_yaw,&mem,&proc_num,dir_name,&bins,rp_dictionary);

    if (y_count > 0) {
      f_yaw = -f_yaw;
      PY_to_PE(&f_pitch,&f_yaw,sum_pitch_vect,&num_yaw,&delta_yaw, \
               pitch_vect,delta_pitch_vect,&rp_per_layer, \
               &proc_num,&mem,&bins);

      load_rp(&f_pitch,&f_yaw,&mem,&proc_num,dir_name,&bins,rp_dictionary);
      f_yaw = -f_yaw;
    }

    f_pitch -= *(delta_pitch_vect+y_count);
  }

  f_yaw += delta_yaw;
}

printf("Do you want to save dictionary of range profiles as a ");
printf("file [1/0]? ");
scanf("%s\n",&want_to_save_dict);

if (want_to_save_dict) {

  /* Save range profile dictionary on disk */
```

```
      printf("Enter dictionary filename? ");
      scanf("%s\n",rp_dict_filename);

      fd = open(rp_dict_filename,O_TRUNC|O_CREAT|O_RDWR,0644);

      if (iproc < (rp_per_layer * bins))
        p_write(fd,rp_dictionary,layers * 4);
      close(fd);

    }
  }
  printf("Done loading dictionary\n");
}
```

```
/*
   Author: Mohammad Faisal

   This is the initialization routine for estimating the orientation of the
   target plane given a range profile representing the data for the
   orientation to be estimated and a given a dictionary of range profiles
   of the target plane. The target plane used is the X29.

   This subroutine can load the individual simulated noiseless range
   profiles generated by the software called XPATCH, and once loaded,
   can save them as a binary file for fast loading.  Whether to lead the
   range profiles individually from XPATCH output files or from pre-stored
   binary file is determined by the variable RP_IN_DICT.

   It opens a socket connection to an SGI machine for displaying the
   estimated orientations. Several other variables are initialized and several
   data structures are computed to be used during the estimation.

   All angles are in degrees.
   Number of range bins in a range profile must be equal or laess than nproc.
   -90 <= yaw <= 90
   -180 <= pitch < 180
   -180 <= roll <= 180
*/

#include <stdio.h>
#include <math.h>
#include <mpl.h>
#include <mpipl.h>
#include "grad_search.h"
#include <stdlib.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/file.h>
#include <sys/types.h>
#include <ppeio.h>
#include "faisalheader.h"

void f_initialize()
{
  extern int close(int);
  visible extern int sendinit_Rome_Iris();
  extern int num_yaw,count,bins,total_num_rp,rp_per_layer,layers, \
      p_count,y_count,mem,proc_num,are_rps_a_file,want_to_save_dict,\
      rp_count,fd;
  extern int *pitch_vect,*sum_pitch_vect,k;
  extern char dir_name[200],data_filename[200],rp_dict_filename[200], \
       dummy_string[200];
  extern float init_pitch,init_yaw,delta_yaw,f_pitch,f_yaw,dummy,val, \
        found_pitch_prime,found_yaw_prime,init_roll,init_pitch_prime, \
        init_yaw_prime,found_pitch,found_yaw;
  extern float *delta_pitch_vect;
  extern float true_pitch,true_yaw,true_roll,true_pitch_prime,true_yaw_prime;
  extern int true_proc_num,true_mem;
  extern plural float *rp_dictionary,*f_data;

  /* Variable for socket comm. */
  extern int f_sock;

  /* Open socket */
```

```
f_sock = callRequest(sendinit_Rome_Iris,4,1989);
        if (f_sock == -1) {
        printf("socket error THESEUS socket error\n");
        exit(1); }
        else printf("THESEUS: socket has been opened\n");

num_yaw = DISCRETIZE_PAR1;

/* Dynamically allocate memory */
pitch_vect = (int *) malloc(num_yaw*sizeof(int));
sum_pitch_vect = (int *) malloc(num_yaw*sizeof(int));
delta_pitch_vect = (float *) malloc(num_yaw*sizeof(int));

delta_yaw = 90.0/(num_yaw-1);

k = DISCRETIZE_PAR2;

for (count=0;count<num_yaw;++count) {
  if (count < (num_yaw-1))
    *(pitch_vect+count) = round(k * cos(delta_yaw * count * 2.0 * PI \
                                        / 360.0));
  else
    *(pitch_vect+count) = 1;
  if (count==0) {
    *sum_pitch_vect = *pitch_vect;
  }
  else {
    *(sum_pitch_vect+count) = *(sum_pitch_vect+count-1) + \
                              *(pitch_vect+count);
  }
  *(delta_pitch_vect+count) = 360.0 / *(pitch_vect+count);
}
total_num_rp = (2 * *(sum_pitch_vect+num_yaw-1)) - *pitch_vect;
bins = NUM_BINS;
are_rps_a_file = RP_IN_DICT;

rp_per_layer = nproc / bins;
layers = ceil(((float) total_num_rp) / ((float) rp_per_layer));

/* Dynamically allocate memory to store range profile dictionary */
rp_dictionary = (plural float *) p_malloc(layers*sizeof(float));

/* loading dictionary in PEs */
if (are_rps_a_file) {

  /* Range profile dictionary already exists as one file on disk.
     Load it in memory */
  sprintf(rp_dict_filename, \
  "/recognition/people/atr/demo/faisalsrc/data/x29_data/x29_dict.float");
  fd = open(rp_dict_filename,O_RDONLY,0);
  if (iproc < (rp_per_layer * bins))
    p_read(fd,rp_dictionary,layers * 4);
  close(fd);
}
else {

  /* Range profile dictionary does not exist as one file on disk.
     Load them one by one using the range profile file naming convention */
  printf("Enter directory name for dictionary? ");
  scanf("%s\n",dir_name);
```

```
count = 0;
f_yaw = 0.0;
for (y_count=0;y_count<num_yaw;++y_count) {

  f_pitch = 0.0;
  for (p_count=0;p_count<((*(pitch_vect+y_count)+1)/2);++p_count) {

    PY_to_PE(&f_pitch,&f_yaw,sum_pitch_vect,&num_yaw,&delta_yaw, \
             pitch_vect,delta_pitch_vect,&rp_per_layer, \
             &proc_num,&mem,&bins);
    load_rp(&f_pitch,&f_yaw,&mem,&proc_num,dir_name,&bins,rp_dictionary);

    if (y_count > 0) {
      f_yaw = -f_yaw;
      PY_to_PE(&f_pitch,&f_yaw,sum_pitch_vect,&num_yaw,&delta_yaw, \
               pitch_vect,delta_pitch_vect,&rp_per_layer, \
               &proc_num,&mem,&bins);
      load_rp(&f_pitch,&f_yaw,&mem,&proc_num,dir_name,&bins,rp_dictionary);
      f_yaw = -f_yaw;
    }

    f_pitch += *(delta_pitch_vect+y_count);
  }

  f_pitch = -(*(delta_pitch_vect+y_count));
  for (p_count=0;p_count<(*(pitch_vect+y_count)/2);++p_count) {

    PY_to_PE(&f_pitch,&f_yaw,sum_pitch_vect,&num_yaw,&delta_yaw, \
             pitch_vect,delta_pitch_vect,&rp_per_layer, \
             &proc_num,&mem,&bins);

    load_rp(&f_pitch,&f_yaw,&mem,&proc_num,dir_name,&bins,rp_dictionary);

    if (y_count > 0) {
      f_yaw = -f_yaw;
      PY_to_PE(&f_pitch,&f_yaw,sum_pitch_vect,&num_yaw,&delta_yaw, \
               pitch_vect,delta_pitch_vect,&rp_per_layer, \
               &proc_num,&mem,&bins);

      load_rp(&f_pitch,&f_yaw,&mem,&proc_num,dir_name,&bins,rp_dictionary);
      f_yaw = -f_yaw;
    }

    f_pitch -= *(delta_pitch_vect+y_count);
  }

  f_yaw += delta_yaw;
}

printf("Do you want to save dictionary of range profiles as a ");
printf("file [1/0]? ");
scanf("%s\n",&want_to_save_dict);

if (want_to_save_dict) {

  /* Save range profile dictionary on disk */
  printf("Enter dictionary filename? ");
  scanf("%s\n",rp_dict_filename);
```

```
      fd = open(rp_dict_filename,O_TRUNC|O_CREAT|O_RDWR,0644);

      if (iproc < (rp_per_layer * bins))
        p_write(fd,rp_dictionary,layers * 4);
      close(fd);

    }
  }
}
```

```
/*
   Author: Mohammad Faisal

   Given a pitch angle for the orientation of the plane, this routine
   returns an angle representing the given angle but within the range
   -180 <= pitch < 180.
*/

#include <stdio.h>
#include <math.h>
#include "faisalheader.h"

void fix_pitch_angle(pitch)

  float *pitch;

{
  if (*pitch < -180.0) {
    *pitch = fmod(*pitch,360.0);
    if (*pitch < -180.0)
      *pitch += 360.0;
  }
  else
    if (*pitch >= 180.0) {
      *pitch = fmod(*pitch,360.0);
      if (*pitch >= 180.0)
        *pitch -= 360.0;
    }
}
```

```
/*
   Author: Mohammad Faisal

   Given the 2 angle orientation of the plane and the roll (3 angle), this
   routine returns the pitch (3 angle).
*/

#include <stdio.h>
#include <math.h>
#include "grad_search.h"
#include "faisalheader.h"

float get_pitch(yaw_prime,pitch_prime,roll)

    float *yaw_prime,*pitch_prime,*roll;

{
    float t1,t2,t3,t4,t5,t6,t7,t8,num,den,p;

    t1 = sin(*roll * D_to_R_scale);
    t2 = sin(*yaw_prime * D_to_R_scale);
    t3 = cos(*roll * D_to_R_scale);
    t4 = sin(*pitch_prime * D_to_R_scale);
    t5 = cos(*yaw_prime * D_to_R_scale);

    t6 = cos(get_yaw(yaw_prime,pitch_prime,roll) * D_to_R_scale);

    num = ((-(t1) * t2) + (t3 * t4 * t5))/t6;

    t7 = cos(*pitch_prime * D_to_R_scale);
    t8 = cos(*yaw_prime * D_to_R_scale);

    den = (t7 * t8)/t6;

    p = atan2(num,den) * R_to_D_scale;
    fix_pitch_angle(&p);
    return(p);
}
```

```
/*
   Author: Mohammad Faisal

   Given the 3 angle orientation of the plane, this routine returns the
   pitch (2 angle).
*/

#include <stdio.h>
#include <math.h>
#include "grad_search.h"
#include "faisalheader.h"

float get_pitch_prime(yaw,pitch,roll)

    float *yaw,*pitch,*roll;

{
    float num,den,t1,t2,t3,t4,t5,t6,t7,t8,pitch_p;

    t1 = sin(*roll * D_to_R_scale);
    t2 = sin(*yaw * D_to_R_scale);
    t3 = cos(*roll * D_to_R_scale);
    t4 = sin(*pitch * D_to_R_scale);
    t5 = cos(*yaw * D_to_R_scale);
    t8 = cos(get_yaw_prime(yaw,pitch,roll) * D_to_R_scale);

    num = ((t1 * t2) + (t3 * t4 * t5))/t8;

    t6 = cos(*pitch * D_to_R_scale);
    t7 = cos(*yaw * D_to_R_scale);

    den = (t6 * t7) / t8;

    pitch_p = atan2(num,den) * R_to_D_scale;
    fix_pitch_angle(&pitch_p);

    return(pitch_p);
}
```

```
/*
   Author: Mohammad Faisal

   Given the 2 angle orientation of the plane and the roll (3 angle), this
   routine returns the pitch (3 angle).
*/

#include <stdio.h>
#include <math.h>
#include "grad_search.h"
#include "faisalheader.h"

float get_yaw(yaw_prime,pitch_prime,roll)

    float *yaw_prime,*pitch_prime,*roll;

{
    float t1,t2,t3,t4,t5;

    t1 = cos(*roll * D_to_R_scale);
    t2 = sin(*yaw_prime * D_to_R_scale);
    t3 = sin(*roll * D_to_R_scale);
    t4 = sin(*pitch_prime * D_to_R_scale);
    t5 = cos(*yaw_prime * D_to_R_scale);

    return(asin((t1 * t2) + (t3 * t4 * t5)) * R_to_D_scale);
}
```

```
/*
   Author: Mohammad Faisal

   Given the 3 angle orientation of the plane, this routine returns the
   yaw (2 angle).
*/

#include <stdio.h>
#include <math.h>
#include "grad_search.h"
#include "faisalheader.h"

float get_yaw_prime(yaw,pitch,roll)

    float *yaw,*pitch,*roll;

{
    float t1,t2,t3,t4,t5;

    t1 = cos(*roll * D_to_R_scale);
    t2 = sin(*yaw * D_to_R_scale);
    t3 = sin(*roll * D_to_R_scale);
    t4 = sin(*pitch * D_to_R_scale);
    t5 = cos(*yaw * D_to_R_scale);

    return(asin((t1 * t2) - (t3 * t4 * t5)) * R_to_D_scale);
}
```

```
/*
   Author: Mohammad Faisal

   This routine loads individual range profiles from files generated by
   the software XPATCH and stores them in the PE array.  A pre-defined
   file naming convention is assumed for the XPATCH range profile files.
*/

#include <stdio.h>
#include <string.h>
#include <math.h>
#include "grad_search.h"
#include <mpl.h>
#include "faisalheader.h"

void load_rp(pitch,yaw,mem,proc_num,dir_name,bins,rp_dictionary)

  float *pitch,*yaw;
  int *mem,*proc_num,*bins;
  char *dir_name;
  plural float *rp_dictionary;

{
  char *filename,*filename_temp,*temp,*dir_name_temp;
  FILE *fp;
  float dummy,pitch_3dp,yaw_3dp;
  float val;
  int count;

  filename = (char *) malloc(200);
  filename_temp = (char *) malloc(200);
  temp = (char *) malloc(100);
  dir_name_temp = (char *) malloc(200);

  pitch_3dp = *pitch - fmod(*pitch,0.001);
  if (pitch_3dp > 0.0)
    sprintf(temp,"/Pp%.3f",pitch_3dp);
  if (pitch_3dp < 0.0)
    sprintf(temp,"/Pm%.3f",fabs(pitch_3dp));
  if (pitch_3dp == 0.0)
    sprintf(temp,"/Pz%.3f",pitch_3dp);
  strcpy(dir_name_temp,dir_name);
  filename_temp = strcat(dir_name_temp,temp);

  yaw_3dp = *yaw - fmod(*yaw,0.001);
  if (yaw_3dp > 0.0)
    sprintf(temp,"_Yp%.3f.range",yaw_3dp);
  if (yaw_3dp < 0.0)
    sprintf(temp,"_Ym%.3f.range",fabs(yaw_3dp));
  if (yaw_3dp == 0.0)
    sprintf(temp,"_Yz%.3f.range",yaw_3dp);
  filename = strcat(filename_temp,temp);

  fp = fopen(filename,"r");
  if (fp == NULL) {
    printf("COULD NOT OPEN %s\n",filename);
    exit(1);
  }

  for (count=0;count<SKIP;++count) {
    fscanf(fp,"%s\n",filename_temp);
  }

  for (count=0;count<*bins;++count) {
    fscanf(fp,"%f %f %f %f %f %f\n",&dummy,&dummy,&dummy,&val,&dummy,&dummy);
    proc[*proc_num].rp_dictionary[*mem] = pow(10.0,(val/10.0));
    ++(*proc_num);
  }

  fclose(fp);

  free(filename);
  free(filename_temp);
  free(temp);
  free(dir_name_temp);
}
```

```
/*
   Author: Mohammad Faisal

   This routine returns the minimum of the two given float numbers.
*/

#include <stdio.h>
#include "faisalheader.h"

float min2(x1,x2)

  float *x1,*x2;

{
  float min;

  min = *x1;
  if (*x2 < min)
    min = *x2;
  return(min);
}
```

```
/*
   Author: Mohammad Faisal

   Given an arbitrary point on the sphere, this routine returns the 2 angle
   nearest grid point.  The nearest grid point is one which, with the given
   arbitrary point, subtends the smallest arc angle.

   In this routine variable names containing pitch implies the elevation of the
   sphere and yaw implies the azimuth of the sphere which should not be
   confused with the pitch and yaw angles of the target plane.
*/

#include <stdio.h>
#include <math.h>
#include "faisalheader.h"

void nearest_rp(float pitch,float yaw,float *nearest_pitch, \
                float *nearest_yaw,int *num_pitch, \
        float *delta_yaw_vect, float *delta_pitch)

{
  struct yaw_node {
    float pitch;
    float left_yaw;
    float right_yaw;
    float left_arc;
    float right_arc;
  };
  struct yaw_node *yaw_node_ptr;
  int pitch_sign,yaw_sign,low_pitch_index,high_pitch_index;
  float min_arc;

  if (pitch < 0.0)
    pitch_sign = -1;
  else
    pitch_sign = 1;

  if (yaw < 0.0)
    yaw_sign = -1;
  else
    yaw_sign = 1;

  pitch = fabs(pitch);
  yaw = fabs(yaw);

  yaw_node_ptr = (struct yaw_node *) malloc(*num_pitch * \
                 sizeof(struct yaw_node));

  /* first find 2 yaw points above and 2 below */
  low_pitch_index = round((pitch-fmod(pitch,*delta_pitch))/ \
                    *delta_pitch);

  if (low_pitch_index < (*num_pitch - 1))
    high_pitch_index = low_pitch_index + 1;
  else
    high_pitch_index = low_pitch_index;

  yaw_node_ptr[low_pitch_index].pitch = low_pitch_index * *delta_pitch;
  yaw_node_ptr[low_pitch_index].left_yaw = yaw-fmod(yaw, \
    delta_yaw_vect[low_pitch_index]);
```

```
yaw_node_ptr[low_pitch_index].right_yaw = \
  yaw_node_ptr[low_pitch_index].left_yaw + delta_yaw_vect[low_pitch_index];
yaw_node_ptr[low_pitch_index].left_arc = \
  arc_angle(&pitch,&yaw,&(yaw_node_ptr[low_pitch_index].pitch), \
            &(yaw_node_ptr[low_pitch_index].left_yaw));
yaw_node_ptr[low_pitch_index].right_arc = \
  arc_angle(&pitch,&yaw,&(yaw_node_ptr[low_pitch_index].pitch), \
            &(yaw_node_ptr[low_pitch_index].right_yaw));

yaw_node_ptr[high_pitch_index].pitch = high_pitch_index * \
                                       *delta_pitch;
yaw_node_ptr[high_pitch_index].left_yaw = yaw-fmod(yaw, \
  delta_yaw_vect[high_pitch_index]);
yaw_node_ptr[high_pitch_index].right_yaw = \
  yaw_node_ptr[high_pitch_index].left_yaw + \
  delta_yaw_vect[high_pitch_index];
yaw_node_ptr[high_pitch_index].left_arc = \
  arc_angle(&pitch,&yaw,&(yaw_node_ptr[high_pitch_index].pitch), \
            &(yaw_node_ptr[high_pitch_index].left_yaw));
yaw_node_ptr[high_pitch_index].right_arc = \
  arc_angle(&pitch,&yaw,&(yaw_node_ptr[high_pitch_index].pitch), \
            &(yaw_node_ptr[high_pitch_index].right_yaw));

/* find the closest of the four yaw points */
min_arc = yaw_node_ptr[low_pitch_index].left_arc;
*nearest_pitch = yaw_node_ptr[low_pitch_index].pitch;
*nearest_yaw = yaw_node_ptr[low_pitch_index].left_yaw;
if (min_arc > yaw_node_ptr[low_pitch_index].right_arc) {
  min_arc = yaw_node_ptr[low_pitch_index].right_arc;
  *nearest_yaw = yaw_node_ptr[low_pitch_index].right_yaw;
}
if (min_arc > yaw_node_ptr[high_pitch_index].left_arc) {
  min_arc = yaw_node_ptr[high_pitch_index].left_arc;
  *nearest_pitch = yaw_node_ptr[high_pitch_index].pitch;
  *nearest_yaw = yaw_node_ptr[high_pitch_index].left_yaw;
}
if (min_arc > yaw_node_ptr[high_pitch_index].right_arc) {
  min_arc = yaw_node_ptr[high_pitch_index].right_arc;
  *nearest_pitch = yaw_node_ptr[high_pitch_index].pitch;
  *nearest_yaw = yaw_node_ptr[high_pitch_index].right_yaw;
}

/* search more yaw point pairs for increasing pitch angles */
while ((high_pitch_index < (*num_pitch - 1)) && \
  (((yaw_node_ptr[high_pitch_index].pitch - pitch) + *delta_pitch) < \
  min_arc)) {

  ++high_pitch_index;
  yaw_node_ptr[high_pitch_index].pitch = high_pitch_index * \
                                         *delta_pitch;
  yaw_node_ptr[high_pitch_index].left_yaw = yaw-fmod(yaw, \
    delta_yaw_vect[high_pitch_index]);
  yaw_node_ptr[high_pitch_index].right_yaw = \
    yaw_node_ptr[high_pitch_index].left_yaw + \
    delta_yaw_vect[high_pitch_index];
  yaw_node_ptr[high_pitch_index].left_arc = \
    arc_angle(&pitch,&yaw,&(yaw_node_ptr[high_pitch_index].pitch), \
              &(yaw_node_ptr[high_pitch_index].left_yaw));
  yaw_node_ptr[high_pitch_index].right_arc = \
    arc_angle(&pitch,&yaw,&(yaw_node_ptr[high_pitch_index].pitch), \
             &(yaw_node_ptr[high_pitch_index].right_yaw));

  if (min_arc < min2(&(yaw_node_ptr[high_pitch_index].left_arc), \
                     &(yaw_node_ptr[high_pitch_index].right_arc)))
    break;
  else {
    min_arc = yaw_node_ptr[high_pitch_index].left_arc;
    *nearest_pitch = yaw_node_ptr[high_pitch_index].pitch;
    *nearest_yaw = yaw_node_ptr[high_pitch_index].left_yaw;
    if (min_arc > yaw_node_ptr[high_pitch_index].right_arc) {
      min_arc = yaw_node_ptr[high_pitch_index].right_arc;
      *nearest_yaw = yaw_node_ptr[high_pitch_index].right_yaw;
    }
  }
}

/* search more yaw points pairs for decreasing pitch angles */
while ((low_pitch_index > 0) && \
  (((pitch - yaw_node_ptr[low_pitch_index].pitch) + *delta_pitch) < \
  min_arc)) {

  --low_pitch_index;
  yaw_node_ptr[low_pitch_index].pitch = low_pitch_index * \
                                         *delta_pitch;
  yaw_node_ptr[low_pitch_index].left_yaw = yaw-fmod(yaw, \
    delta_yaw_vect[low_pitch_index]);
  yaw_node_ptr[low_pitch_index].right_yaw = \
    yaw_node_ptr[low_pitch_index].left_yaw + \
    delta_yaw_vect[low_pitch_index];
  yaw_node_ptr[low_pitch_index].left_arc = \
    arc_angle(&pitch,&yaw,&(yaw_node_ptr[low_pitch_index].pitch), \
              &(yaw_node_ptr[low_pitch_index].left_yaw));
  yaw_node_ptr[low_pitch_index].right_arc = \
    arc_angle(&pitch,&yaw,&(yaw_node_ptr[low_pitch_index].pitch), \
              &(yaw_node_ptr[low_pitch_index].right_yaw));

  if (min_arc < min2(&(yaw_node_ptr[low_pitch_index].left_arc), \
                     &(yaw_node_ptr[low_pitch_index].right_arc)))
    break;
  else {
    min_arc = yaw_node_ptr[low_pitch_index].left_arc;
    *nearest_pitch = yaw_node_ptr[low_pitch_index].pitch;
    *nearest_yaw = yaw_node_ptr[low_pitch_index].left_yaw;
    if (min_arc > yaw_node_ptr[low_pitch_index].right_arc) {
      min_arc = yaw_node_ptr[low_pitch_index].right_arc;
      *nearest_yaw = yaw_node_ptr[low_pitch_index].right_yaw;
    }
  }
}


/* take care of the signs for pitch and yaw */
*nearest_pitch *= pitch_sign;
*nearest_yaw *= yaw_sign;

/* take care of yaw = 180 .. change it to -180 */
/* sometimes yaw comes out to be 360 or -360 rather than 0 so fix that */
fix_pitch_angle(nearest_yaw);

free(yaw_node_ptr);
```

```
/*
   Author: Mohammad Faisal

   Given a point on the unit sphere, this routine computes the cartesian
   coordinates of that point.

   In this routine variable names containing pitch implies the elevation of the
   sphere and yaw implies the azimuth of the sphere which should not be
   confused with the pitch and yaw angles of the target plane.
*/

#include <stdio.h>
#include <math.h>
#include "grad_search.h"
#include "faisalheader.h"

void polar_to_cart(pitch,yaw,x,y,z)

  float *pitch,*yaw,*x,*y,*z;

{
  float p_radians,y_radians;

  p_radians = *pitch * 2 * PI / 360.0;
  y_radians = *yaw * 2 * PI / 360.0;
  *x = cos(p_radians) * cos(y_radians);
  *y = cos(p_radians) * sin(y_radians);
  *z = sin(p_radians);
}
```

```
/*
   Author: Mohammad Faisal

   Given a floating point number, this routine returns the nearest rounded
   integer.
*/

#include <stdio.h>
#include <math.h>
#include "faisalheader.h"

int round(float x)

{
  if ((x - floor(x)) < 0.5)
    return(floor(x));
  else
    return(ceil(x));
}
```

```
/*
   Author: Mohammad Faisal

   Given a grid point on the sphere and the range profile data, this routine
   computes the mean squared error between the range profile data and the
   dictionary range profile corresponding to the grid point.
*/

#include <stdio.h>
#include <mpl.h>
#include "faisalheader.h"

float rp_diff(pitch,yaw,rp_dictionary,f_data,sum_pitch_vect,num_yaw, \
              delta_yaw,pitch_vect,delta_pitch_vect,rp_per_layer, \
              bins)

  float *pitch,*yaw,*delta_yaw,*delta_pitch_vect;
  plural float *rp_dictionary,*f_data;
  int *sum_pitch_vect,*num_yaw,*pitch_vect,*rp_per_layer,*bins;

{
  int proc_num,mem;
  plural float sq_d;

  PY_to_PE(pitch,yaw,sum_pitch_vect,num_yaw,delta_yaw,pitch_vect, \
           delta_pitch_vect,rp_per_layer,&proc_num,&mem,bins);

  if ((iproc >= proc_num) && (iproc < (proc_num + *bins))) {
    sq_d = *f_data - rp_dictionary[mem];
    sq_d *= sq_d;

    return(reduceAddf(sq_d));
  }
}
```

```
/*
   Author: Mohammad Faisal

   This routine uses an initial guess of the orientation of the target plane
   (in 3 angles) and calls another routine to perform gradient based search
   to estimate the orientation of the target plane using range profile data
   and range profile dictionary.  The initial guess of the orientation is
   also sent over the socket to the SGI for visualization.

   All angles are in degrees.
   Number of range bins in a range profile must be equal or laess than nproc.
   -90 <= yaw <= 90
   -180 <= pitch < 180
   -180 <= roll <= 180
*/

#include <stdio.h>
#include <math.h>
#include <mpl.h>
#include <mpipl.h>
#include "grad_search.h"
#include <stdlib.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/file.h>
#include <sys/types.h>
#include <ppeio.h>
#include "faisalheader.h"

void search()
{
  extern int close(int);
  visible extern int sendinit_Rome_Iris();
  extern int num_yaw,count,bins,total_num_rp,rp_per_layer,layers, \
      p_count,y_count,mem,proc_num,are_rps_a_file,want_to_save_dict,\
      rp_count,fd;
  extern int *pitch_vect,*sum_pitch_vect,k;
  extern char dir_name[200],data_filename[200],rp_dict_filename[200], \
       dummy_string[200];
  extern float init_pitch,init_yaw,delta_yaw,f_pitch,f_yaw,dummy,val, \
        found_pitch_prime,found_yaw_prime,init_roll,init_pitch_prime, \
        init_yaw_prime,found_pitch,found_yaw, found_roll;
  extern float *delta_pitch_vect,typ,tpp,flag,z1,z2;
  extern float true_pitch,true_yaw,true_roll,true_pitch_prime,true_yaw_prime;
  extern int true_proc_num,true_mem;
  extern unsigned char test_flag;
  extern plural float *rp_dictionary,*f_data,noise_re,noise_im;
  extern FILE *fp;

  /* Variable for socket comm. */
  extern int f_sock;

  /* send initial guess of orientation over the socket to SGI */
  write(f_sock, &test_flag, 1);
  write(f_sock, &init_pitch, 4);
  write(f_sock, &init_yaw, 4);
  write(f_sock, &init_roll, 4);

  /* reduce initial guess of orientation from 3 angles to 2 angles */
  init_yaw_prime = get_yaw_prime(&init_yaw,&init_pitch,&init_roll);
  init_pitch_prime = get_pitch_prime(&init_yaw,&init_pitch,&init_roll);

  /* Gradient search begins */
  trace(f_data,rp_dictionary,&init_pitch_prime,&init_yaw_prime, \
        &found_pitch_prime, \
        &found_yaw_prime,&bins,&delta_yaw,&num_yaw,delta_pitch_vect, \
        sum_pitch_vect,pitch_vect,&rp_per_layer,f_sock,&init_roll);

  /* expand estimated orientation from 2 angles to 3 angles */
  found_pitch = get_pitch(&found_yaw_prime,&found_pitch_prime, \
                          &init_roll);
  found_yaw = get_yaw(&found_yaw_prime,&found_pitch_prime, \
                      &init_roll);
  found_roll = init_roll;
}
```

```
/*
   Author: Mohammad Faisal

   This routine locates in PE memory the dictionary range profile corresponding
   to the true orientation of the plane and sends it over the socket to the
   SGI for visualization.  It also uses this range profile as the data
   range profile which is used to estimate the true orientation of the target.
   Since the true orientation of the target plane may not lie on a grid point
   within the 2 angle orientations space (true orientation is specified in
   3 angles and then reduced to 2 angles) the nearest grid point is selected.

   All angles are in degrees.
   Number of range bins in a range profile must be equal or laess than nproc.
   -90 <= yaw <= 90
   -180 <= pitch < 180
   -180 <= roll <= 180
*/

#include <stdio.h>
#include <math.h>
#include <mpl.h>
#include <mpipl.h>
#include "grad_search.h"
#include <stdlib.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/file.h>
#include <sys/types.h>
#include <ppeio.h>
#include "faisalheader.h"
#include "../include/param.h"
#include "../include/main.h"

void simulate(int first_time)
{
  extern int close(int);
  visible extern int sendinit_Rome_Iris();
  extern int num_yaw,count,bins,total_num_rp,rp_per_layer,layers, \
      p_count,y_count,mem,proc_num,are_rps_a_file,want_to_save_dict,\
      rp_count,fd;
  extern int *pitch_vect,*sum_pitch_vect,k;
  extern char dir_name[200],data_filename[200],rp_dict_filename[200], \
       dummy_string[200];
  extern float init_pitch,init_yaw,delta_yaw,f_pitch,f_yaw,dummy,val, \
        found_pitch_prime,found_yaw_prime,init_roll,init_pitch_prime, \
        init_yaw_prime,found_pitch,found_yaw;
  extern float *delta_pitch_vect,typ,tpp,flag,z1,z2;
  extern float true_pitch,true_yaw,true_roll,true_pitch_prime,true_yaw_prime;
  extern int true_proc_num,true_mem;
  extern plural float *rp_dictionary,*f_data;
  extern FILE *fp;

  /* Variable for socket comm. */
  extern int f_sock;

  unsigned char newflag;

  /* print the true orientation angles of the plane for which range profile
     data is to be simulated */
  printf("SIM: TRUTH: %0.3f %0.3f %0.3f\n",true_pitch,true_yaw,true_roll);
  /* send true orientation over the socket */
  newflag = 5;

  /* Sending True Orientations over the sokcet to the SGI for visualization */
  printf("Sending True Orientations over the sokcet \n");
  write(f_sock, &newflag, 1);
  write(f_sock, &true_pitch, 4);
  write(f_sock, &true_yaw, 4);
  write(f_sock, &true_roll, 4);

  /* sumulating data */
  printf("Simulating data ...\n");
  f_data = (plural float *) p_malloc(sizeof(float));

  typ = get_yaw_prime(&true_yaw,&true_pitch,&true_roll);
  tpp = get_pitch_prime(&true_yaw,&true_pitch,&true_roll);
  printf("Truth in 2 angles:[pitch yaw] = %f %f\n",tpp,typ);

  /* find nearest grid point in the 2 angle orientation space corresponding
     to the 2 angle true orientation */
  nearest_rp(typ,tpp,&true_yaw_prime,&true_pitch_prime, \
             &num_yaw,delta_pitch_vect,&delta_yaw);
  printf("Grid point nearest truth:[pitch yaw] = %f %f\n", \
         true_pitch_prime,true_yaw_prime);
  printf("Three angle conversion: [pitch yaw roll] = %f %f %f\n",
         get_pitch(&true_yaw_prime,&true_pitch_prime,&true_roll),
         get_yaw(&true_yaw_prime,&true_pitch_prime,&true_roll),true_roll);


  /* locate the PE and memory depth where corresponding dictionary range
     profile is located */
  PY_to_PE(&true_pitch_prime,&true_yaw_prime,sum_pitch_vect,&num_yaw,&delta_yaw,
\
           pitch_vect,delta_pitch_vect,&rp_per_layer,&true_proc_num, \
           &true_mem,&bins);
  printf("True range profile locator:[proc_num depth] %d %d\n", \
                true_proc_num,true_mem);


  write(f_sock, &newflag, 1);
  for (count=0;count<bins;++count) {
    val = proc[true_proc_num+count].rp_dictionary[true_mem];

    /* send noiseless range profile corresponding to nearest grid point
       to the true orientation
       over the socket */

    write(f_sock, &val, 4);

    for(rp_count=0;rp_count<rp_per_layer;++rp_count) {
      proc[count+(rp_count * bins)].f_data[0] = val;
    }
  }
}
```

```
/*
   Author: Mohammad Faisal

   Given a pair of points on the sphere, this routine returns the gradient
   of the likelihiid surface of the sphere interpolated between the two
   points.  The chord length between the two points is used to divide the
   difference of the surface values at the two points to compute the
   gradient.
*/

#include <stdio.h>
#include <math.h>
#include <mpl.h>
#include "faisalheader.h"

float slope(pitch1,yaw1,diff,pitch2,yaw2, \
            rp_dictionary,f_data,sum_pitch_vect,num_yaw,delta_yaw,
            pitch_vect,delta_pitch_vect,rp_per_layer,bins)

  float *pitch1,*yaw1,*diff,*pitch2,*yaw2;
  float plural *rp_dictionary,*f_data;
  float *delta_yaw,*delta_pitch_vect;
  int *sum_pitch_vect,*num_yaw,*pitch_vect,*rp_per_layer,*bins;

{
  float diff2,chord;

  diff2 = rp_diff(pitch2,yaw2,rp_dictionary,f_data,sum_pitch_vect, \
                  num_yaw,delta_yaw,pitch_vect,delta_pitch_vect, \
                  rp_per_layer,bins);
  chord = sqrt(sq_chord(yaw1,pitch1,yaw2,pitch2));

  return((diff2 - *diff)/chord);
}
```

```
/*
   Author: Mohammad Faisal

   Given a pair of points on the sphere, this routine returns the chord length
   between the two points assuming a unit radius sphere.

   In this routine variable names containing pitch implies the elevation of the
   sphere and yaw implies the azimuth of the sphere which should not be
   confused with the pitch and yaw angles of the target plane.
*/

#include <stdio.h>
#include "faisalheader.h"

float sq_chord(pitch1,yaw1,pitch2,yaw2)

  float *pitch1,*yaw1,*pitch2,*yaw2;

{
  float x1,y1,z1,x2,y2,z2,d1,d2,d3;

  polar_to_cart(pitch1,yaw1,&x1,&y1,&z1);
  polar_to_cart(pitch2,yaw2,&x2,&y2,&z2);

  d1 = x1-x2;
  d2 = y1-y2;
  d3 = z1-z2;

  return((d1 * d1)+(d2 * d2)+(d3 * d3));
}
```

```
/*
   Author: Mohammad Faisal

   This routine performs the gradient based search given a range profile
   dictionary, range profile data corresponding to the target plane, and
   an initial guess to the orientation of the target plane to estimate the
   true orientation.  The 2 angle surface over which the search is performed
   is a sphere, whose elevation represents the 2 angle yaw and whose azimuth
   represents the 2 angle pitch.  A fixed number of gradient based diffusions
   are performed determined by the variable NUM_DIFF.  The sphere is
   discretized and each point corresponds to a 2 angle orientation with an
   associated surface value equal to the mean square error between the
   noiseless dictionary range profile corresponding to the point and the
   range profile data.

   All angles are in degrees.
   Number of range bins in a range profile must be equal or laess than nproc.
   -90 <= yaw <= 90
   -180 <= pitch < 180
   -180 <= roll <= 180
*/

#include <stdio.h>
#include <math.h>
#include <mpl.h>
#include "grad_search.h"
#include "faisalheader.h"

void trace(f_data,rp_dictionary,init_pitch,init_yaw,found_pitch, \
           found_yaw,bins,delta_yaw,num_yaw,delta_pitch_vect, \
           sum_pitch_vect,pitch_vect,rp_per_layer,f_sock,expanded_roll)

  float plural *f_data,*rp_dictionary;
  float *found_pitch,*found_yaw,*rp_per_layer;
  float *init_pitch,*init_yaw,*delta_yaw,*delta_pitch_vect;
  int *num_yaw,*sum_pitch_vect,*pitch_vect,f_sock,*bins;
  float *expanded_roll;

{
  float left_pitch,left_yaw,right_pitch,right_yaw, \
        top_pitch,top_yaw,bottom_pitch,bottom_yaw,diff,min_slope, \
        next_pitch,next_yaw,next_slope,expanded_yaw,expanded_pitch;
  int yaw_index,count,proc_num,mem,diff_count;
  float val;
  FILE *fp;
  unsigned char test_flag = 5;

  /* find nearest grid point in the orientation space to the initial guess
     of the target plane */
  nearest_rp(*init_yaw,*init_pitch,found_yaw,found_pitch,num_yaw, \
             delta_pitch_vect,delta_yaw);

  /* expand the orientation corresponding to the nearest grid point from
     2 angles to 3 angles */
  expanded_yaw = get_yaw(found_yaw,found_pitch,expanded_roll);
  expanded_pitch = get_pitch(found_yaw,found_pitch,expanded_roll);

  diff = rp_diff(found_pitch,found_yaw,rp_dictionary,f_data, \
                 sum_pitch_vect,num_yaw,delta_yaw,pitch_vect, \
                 delta_pitch_vect,rp_per_layer,bins);
```

```
/* locate the PE and memory depth where corresponding dictionary range
   profile is located */
PY_to_PE(found_pitch,found_yaw,sum_pitch_vect,num_yaw,delta_yaw, \
         pitch_vect,delta_pitch_vect,rp_per_layer,&proc_num,&mem,bins);

/* send 3 angle orientation corresponding to nearest grid point
   to the initial guess via socket to SGI.  This orientation is displayed
   as corresponding to the initial guess  since we don't have the
   range profile corresponding to the initial guess */
write(f_sock, &test_flag, 1);
write(f_sock, &expanded_pitch, 4);
write(f_sock, &expanded_yaw, 4);
write(f_sock, expanded_roll, 4);

/* perform gradient based search using fixed number of diffusions */
for (diff_count = 0;diff_count < NUM_DIFF; diff_count++) {

  /* if not at pole of sphere*/
  if ((*found_yaw != 90.0) && (*found_yaw != -90.0)) {
    left_yaw = right_yaw = *found_yaw;
    yaw_index = round(fabs(*found_yaw)/ *delta_yaw);
    left_pitch = *found_pitch - delta_pitch_vect[yaw_index];
    fix_pitch_angle(&left_pitch);
    right_pitch = *found_pitch + delta_pitch_vect[yaw_index];
    fix_pitch_angle(&right_pitch);

    top_yaw = *found_yaw + *delta_yaw;
    yaw_index = round(fabs(top_yaw)/ *delta_yaw);
    top_pitch = round(*found_pitch/delta_pitch_vect[yaw_index]) \
                * delta_pitch_vect[yaw_index];
    fix_pitch_angle(&top_pitch);

    bottom_yaw = *found_yaw - *delta_yaw;
    yaw_index = round(fabs(bottom_yaw)/ *delta_yaw);
    bottom_pitch = round(*found_pitch/delta_pitch_vect[yaw_index]) \
                   * delta_pitch_vect[yaw_index];
    fix_pitch_angle(&bottom_pitch);

    min_slope = slope(found_pitch,found_yaw,&diff,&top_pitch, \
                      &top_yaw,rp_dictionary,f_data,sum_pitch_vect, \
                      num_yaw,delta_yaw,pitch_vect,delta_pitch_vect, \
                      rp_per_layer,bins);
    next_pitch = top_pitch;
    next_yaw = top_yaw;

    next_slope = slope(found_pitch,found_yaw,&diff,&bottom_pitch, \
                       &bottom_yaw,rp_dictionary,f_data,sum_pitch_vect, \
                       num_yaw,delta_yaw,pitch_vect,delta_pitch_vect, \
                       rp_per_layer,bins);

    if (next_slope < min_slope) {
      min_slope = next_slope;
      next_pitch = bottom_pitch;
      next_yaw = bottom_yaw;
    }

    /* do not compute left gradient if left point is the same as current */
    if (left_pitch != *found_pitch) {
      next_slope = slope(found_pitch,found_yaw,&diff,&left_pitch, \
                            &left_yaw,rp_dictionary,f_data,sum_pitch_vect, \
                            num_yaw,delta_yaw,pitch_vect,delta_pitch_vect, \
                            rp_per_layer,bins);

      if (next_slope < min_slope) {
        min_slope = next_slope;
        next_pitch = left_pitch;
        next_yaw = left_yaw;
      }
    }

    /* do not compute right gradient if right point is the same as current */
    if (right_pitch != *found_pitch) {
      next_slope = slope(found_pitch,found_yaw,&diff,&right_pitch, \
                            &right_yaw,rp_dictionary,f_data,sum_pitch_vect, \
                            num_yaw,delta_yaw,pitch_vect,delta_pitch_vect, \
                            rp_per_layer,bins);

      if (next_slope < min_slope) {
        min_slope = next_slope;
        next_pitch = right_pitch;
        next_yaw = right_yaw;
      }
    }
  }

  /* if at pole of the sphere */
  else {
    next_pitch = right_pitch = 0.0;
    if (*found_yaw == 90.0)
      next_yaw = 90.0 - *delta_yaw;
    else
      next_yaw = -90.0 + *delta_yaw;
    min_slope = slope(found_pitch,found_yaw,&diff,&next_pitch, \
                         &next_yaw,rp_dictionary,f_data,sum_pitch_vect, \
                         num_yaw,delta_yaw,pitch_vect,delta_pitch_vect, \
                         rp_per_layer,bins);

    for (count=0;count<(pitch_vect[*num_yaw-2]-1);++count) {
      right_pitch += delta_pitch_vect[*num_yaw-2];
      fix_pitch_angle(&right_pitch);
      next_slope = slope(found_pitch,found_yaw,&diff,&right_pitch, \
                            &next_yaw,rp_dictionary,f_data,sum_pitch_vect, \
                            num_yaw,delta_yaw,pitch_vect,delta_pitch_vect, \
                            rp_per_layer,bins);
      if (next_slope < min_slope) {
        min_slope = next_slope;
        next_pitch = right_pitch;
      }
    }

  }

  if (min_slope >= 0.0) {

    /* expand the newly selected grid point at the end of the diffusion
       from 2 angles to 3 angles */
    expanded_yaw = get_yaw(found_yaw,found_pitch,expanded_roll);
    expanded_pitch = get_pitch(found_yaw,found_pitch,expanded_roll);
```

```
        /* send this intermediate estimate of the orientation over the socket
           to the SGI for visualization */
        write(f_sock, &test_flag, 1);
        write(f_sock, &expanded_pitch, 4);
        write(f_sock, &expanded_yaw, 4);
        write(f_sock, expanded_roll, 4);

        /* send noiseless range profile corresponding to new estimate
           via socket */
        PY_to_PE(found_pitch,found_yaw,sum_pitch_vect,num_yaw,delta_yaw, \
                 pitch_vect,delta_pitch_vect,rp_per_layer,&proc_num,&mem,bins);
        if (diff_count == NUM_DIFF - 1) {
          write(f_sock, &test_flag, 1);
          for (count=0;count<*bins;++count) {
            val = proc[proc_num+count].rp_dictionary[mem];
            write(f_sock, &val, 4);
          }
        }
      }
      else {
        *found_pitch = next_pitch;
        *found_yaw = next_yaw;
        diff = rp_diff(found_pitch,found_yaw,rp_dictionary,f_data, \
                       sum_pitch_vect,num_yaw,delta_yaw,pitch_vect, \
                       delta_pitch_vect,rp_per_layer,bins);

        /* expand the newly selected grid point at the end of the diffusion
           from 2 angles to 3 angles */
        expanded_yaw = get_yaw(found_yaw,found_pitch,expanded_roll);
        expanded_pitch = get_pitch(found_yaw,found_pitch,expanded_roll);

        /* send this intermediate estimate of the orientation over the socket
           to the SGI for visualization */
        write(f_sock, &test_flag, 1);
        write(f_sock, &expanded_pitch, 4);
        write(f_sock, &expanded_yaw, 4);
        write(f_sock, expanded_roll, 4);

        /* send noiseless range profile corresponding to new estimate
           via socket */
        PY_to_PE(found_pitch,found_yaw,sum_pitch_vect,num_yaw,delta_yaw, \
                 pitch_vect,delta_pitch_vect,rp_per_layer,&proc_num,&mem,bins);
        if (diff_count == NUM_DIFF - 1) {
          write(f_sock, &test_flag, 1);
          for (count=0;count<*bins;++count) {
            val = proc[proc_num+count].rp_dictionary[mem];
            write(f_sock, &val, 4);
          }
        }
      }
    }
  }
}
```

```
#ifdef __STDC__
# define        P(s) s
#else
# define P(s) ()
#endif


/* PY_to_PE.m */
void PY_to_PE P((float *f_pitch , float *f_yaw , int *sum_pitch_vect , int *num_yaw , floa
t *delta_yaw , int *pitch_vect , float *delta_pitch_vect , int *rp_per_layer , int *proc_n
um , int *mem , int *bins ));

/* round.m */
int round P((float x ));

/* load_rp.m */
void load_rp P((float *pitch , float *yaw , int *mem , int *proc_num , char *dir_name , in
t *bins , plural float *rp_dictionary ));

/* arc_angle.m */
float arc_angle P((float *pitch1 , float *yaw1 , float *pitch2 , float *yaw2 ));

/* fix_pitch_angle.m */
void fix_pitch_angle P((float *pitch ));

/* min2.m */
float min2 P((float *x1 , float *x2 ));

/* nearest_rp.m */
void nearest_rp P((float pitch , float yaw , float *nearest_pitch , float *nearest_yaw , i
nt *num_pitch , float *delta_yaw_vect , float *delta_pitch ));

/* polar_to_cart.m */
void polar_to_cart P((float *pitch , float *yaw , float *x , float *y , float *z ));

/* rp_diff.m */
float rp_diff P((float *pitch , float *yaw , plural float *rp_dictionary , plural float *f
_data , int *sum_pitch_vect , int *num_yaw , float *delta_yaw , int *pitch_vect , float *d
elta_pitch_vect , int *rp_per_layer , int *bins ));

/* slope.m */
float slope P((float *pitch1 , float *yaw1 , float *diff , float *pitch2 , float *yaw2 , f
loat plural *rp_dictionary , float plural *f_data , int *sum_pitch_vect , int *num_yaw , f
loat *delta_yaw , int *pitch_vect , float *delta_pitch_vect , int *rp_per_layer , int *bin
s ));

/* sq_chord.m */
float sq_chord P((float *pitch1 , float *yaw1 , float *pitch2 , float *yaw2 ));

/* trace.m */
void trace P((float plural *f_data , float plural *rp_dictionary , float *init_pitch , flo
at *init_yaw , float *found_pitch , float *found_yaw , int *bins , float *delta_yaw , int
*num_yaw , float *delta_pitch_vect , int *sum_pitch_vect , int *pitch_vect , float *rp_per
_layer , int f_sock , float *expanded_roll ));

/* get_yaw_prime.m */
float get_yaw_prime P((float *yaw , float *pitch , float *roll ));

/* get_pitch_prime.m */
float get_pitch_prime P((float *yaw , float *pitch , float *roll ));
```

```
/* get_pitch.m */
float get_pitch P((float *yaw_prime , float *pitch_prime , float *roll ));

/* get_yaw.m */
float get_yaw P((float *yaw_prime , float *pitch_prime , float *roll ));

/* f_initialize.m */
void f_initialize P((void ));

/* simulate.m */
void simulate P((int first_time ));

/* search.m */
void search P((void ));

/* f_init_jupiter.m */
void f_init_jupiter P((void ));

#undef P
```

```
#define SKIP 86
#define PI 3.14159265359
#define R_to_D_scale 57.29578
#define D_to_R_scale 0.017453293
#define DISCRETIZE_PAR1 10
#define DISCRETIZE_PAR2 36
#define NUM_BINS 1024
#define RP_IN_DICT 1
#define NOISE_SCALE 0.0
#define NUM_DIFF 4
#define ITER 3
```

```
float min2();
```

```
/*****************************************************************
**                                                              **
**      OBJ format header file                                  **
**                                                              **
*****************************************************************/

/* $Id: OBJ.h,v 1.6 94/04/26 01:16:30 rst Exp Locker: rst $
 *
 * $Log:        OBJ.h,v $
 * Revision 1.6  94/04/26  01:16:30  rst
 * fixed 80 char line size limitation to 255 char
 *
 * Revision 1.5  1994/03/24  14:49:50  rst
 * obj_type structure added to simplify routine usage
 *
 * Revision 1.4  1994/03/23  11:20:55  rst
 * Updated to follow ANSI C conventions
 *
 * Revision 1.3  93/11/10  11:30:14  rst
 * better comments on how to use the routines
 *
 * Revision 1.2  93/10/27  00:43:55  rst
 * completed header file
 *
 * Revision 1.1  93/10/26  22:21:07  rst
 * Initial revision
 *
 *
 */


/***********************/
/* OBJ format routines */
/***********************/

#define LINESZ 255
#define MAXVERT 100000
#define MAXNORM 100000

/* internal type definitions */

typedef struct corner_list {
        int vertex, normal;
        struct corner_list *next;
        } corner_cell;

typedef struct face_list {
        struct corner_list *face;
        struct face_list *next_face;
        } face_cell;

/* obj object type definition */

typedef struct obj_object {
        face_cell *head;
        float vert[MAXVERT][3];
        float norm[MAXNORM][3];
        float max;
        } obj_type;
```

```
typedef short boolean;

#define TRUE 1
#define FALSE 0

/* end of type defs */

/* generators for face and corner cells */
face_cell *new_face(face_cell **current);
corner_cell *new_vertex(corner_cell **current, int vertex, int normal);

/* read a single line procedure */
int readline(FILE *fl, char line[LINESZ]);

/* read next set of vertex//normal indices */
int readnext(char **line_ptr, int *vertp, int *normp);

/* procedures to calculate normals */
int cross(float v2[3], float v1[3], float u[3]);
int do_normals(face_cell *facep, int *jp, float vert[][3], float norm[][3]);

/***************************************************************************/
/* These are the two procedures that will get called by outside programs */
/***************************************************************************/

/* read data procedure */
/* invoked as:
        obj_read(filename,&object)
   where:
        filename is the name of the .obj file to be read ("f16M.obj")
        object is a structure of type obj_type
*/
int obj_read(char *filenm, obj_type *object);


/* procedure to draw object from list format */
/* invoked as:
        obj_draw(object,shade)
   where:
        object is a structure of type obj_type
        shade is a boolean telling whether to draw shaded (TRUE) or wireframe
*/
int obj_draw(obj_type object, boolean shade);
```

```
#ifdef __STDC__
# define        P(s) s
#else
# define P(s) ()
#endif


/* PY_to_PE.m */
void PY_to_PE P((float *f_pitch , float *f_yaw , int *sum_pitch_vect , int *num_yaw , floa
t *delta_yaw , int *pitch_vect , float *delta_pitch_vect , int *rp_per_layer , int *proc_n
um , int *mem , int *bins ));

/* round.m */
int round P((float x ));

/* load_rp.m */
void load_rp P((float *pitch , float *yaw , int *mem , int *proc_num , char *dir_name , in
t *bins , plural float *rp_dictionary ));

/* arc_angle.m */
float arc_angle P((float *pitch1 , float *yaw1 , float *pitch2 , float *yaw2 ));

/* fix_pitch_angle.m */
void fix_pitch_angle P((float *pitch ));

/* min2.m */
float min2 P((float *x1 , float *x2 ));

/* nearest_rp.m */
void nearest_rp P((float pitch , float yaw , float *nearest_pitch , float *nearest_yaw , i
nt *num_pitch , float *delta_yaw_vect , float *delta_pitch ));

/* polar_to_cart.m */
void polar_to_cart P((float *pitch , float *yaw , float *x , float *y , float *z ));

/* rp_diff.m */
float rp_diff P((float *pitch , float *yaw , plural float *rp_dictionary , plural float *f
_data , int *sum_pitch_vect , int *num_yaw , float *delta_yaw , int *pitch_vect , float *d
elta_pitch_vect , int *rp_per_layer , int *bins ));

/* slope.m */
float slope P((float *pitch1 , float *yaw1 , float *diff , float *pitch2 , float *yaw2 , f
loat plural *rp_dictionary , float plural *f_data , int *sum_pitch_vect , int *num_yaw , f
loat *delta_yaw , int *pitch_vect , float *delta_pitch_vect , int *rp_per_layer , int *bin
s ));

/* sq_chord.m */
float sq_chord P((float *pitch1 , float *yaw1 , float *pitch2 , float *yaw2 ));

/* trace.m */
void trace P((float plural *f_data , float plural *rp_dictionary , float *init_pitch , flo
at *init_yaw , float *found_pitch , float *found_yaw , int *bins , float *delta_yaw , int
*num_yaw , float *delta_pitch_vect , int *sum_pitch_vect , int *pitch_vect , float *rp_per
_layer , int f_sock , float *expanded_roll ));

/* get_yaw_prime.m */
float get_yaw_prime P((float *yaw , float *pitch , float *roll ));

/* get_pitch_prime.m */
float get_pitch_prime P((float *yaw , float *pitch , float *roll ));
```

```
/* get_pitch.m */
float get_pitch P((float *yaw_prime , float *pitch_prime , float *roll ));

/* get_yaw.m */
float get_yaw P((float *yaw_prime , float *pitch_prime , float *roll ));

/* f_initialize.m */
void f_initialize P((void ));

/* simulate.m */
void simulate P((int first_time ));

/* search.m */
void search P((void ));

/* f_init_jupiter.m */
void f_init_jupiter P((void ));

#undef P
```

```
/*****************************************************************
**
**
**      flip file format header file
**
**
**
*****************************************************************/

/* $Id: flip.h,v 1.1 1994/03/24 15:06:49 rst Exp $
 *
 * $Log: flip.h,v $
 * Revision 1.1  1994/03/24  15:06:49  rst
 * Initial revision
 *
 *
 */


/************************/
/* flip format routines */
/************************/

/* type definitions */

typedef struct flip_object {
        float *ptr;
        long size;
        } flip_type;

/* end of type defs */

/*************************************************************************/
/* These are the two procedures that will get called by outside programs */
/*************************************************************************/

/* read data procedure */
/* invoked as:
        flip_read(filename,&object)
   where:
        filename is the name of the flip format file to be read ("plane.bin")
        object is a structure of type flip_type
*/
int flip_read(char *filenm, flip_type *object);


/* procedure to draw object */
/* invoked as:
        flip_draw(object)
   where:
        object is a structure of type flip_type
*/
int flip_draw(flip_type object);
```

```
#define SKIP 86
#define PI 3.14159265359
#define R_to_D_scale 57.29578
#define D_to_R_scale 0.017453293
#define DISCRETIZE_PAR1 10
#define DISCRETIZE_PAR2 36
#define NUM_BINS 1024
#define RP_IN_DICT 1
#define NOISE_SCALE 0.0
#define NUM_DIFF 4
#define ITER 3
```

```
#include <../include/param.h>
#ifdef __STDC__
# define        P(s) s
#else
# define P(s) ()
#endif


/* base.m */
void track_init P((void ));
void base P((plural float expo , plural float uniform_rv ));
int read_sock P((int insock , char *ptr , int sz ));

/* covariance.m */
void form_cov P((int n , plural float cov [OBJNUM_MAX ][3 ][2 ], int obj_ind , float p_rat
e [OBJNUM_MAX ], float q_rate [OBJNUM_MAX ], float r_rate [OBJNUM_MAX ]));
void shift_cov_in P((int obj_ind , plural float A [OBJNUM_MAX ][3 ][2 ]));
void angle_diff P((float x1 , float x2 , float *y ));

/* reshaping_covariance.m */
void reshape_cov P((int n_obj [OBJNUM_MAX ], int sel , plural float cov_in [OBJNUM_MAX ][3
 ][2 ], plural float cov_out [OBJNUM_MAX ][3 ][2 ]));
void shift_cov_out P((int sel , plural float cov_out [OBJNUM_MAX ][3 ][2 ]));

/* jump_process.m */
void jump_metro P((int n_obj [OBJNUM_MAX ], int sel , int read_indx , plural float theta ,
 plural float phi , plural float psi , plural float *u , plural float *v , plural float *w
 , plural float T [3 ][3 ], float p_rate [OBJNUM_MAX ], float q_rate [OBJNUM_MAX ], float
r_rate [OBJNUM_MAX ]));

/* matrix_mult.m */
void mymatmul P((int n , int obj_ind , plural float T [3 ][3 ], plural float cov_out [OBJN
UM_MAX ][3 ][2 ], plural float K_x [OBJNUM_MAX ][3 ][2 ]));
void mymatmul2 P((int n , plural float T [3 ][3 ], plural float cov_out [3 ][3 ], plural f
loat K_x [3 ][3 ]));
void mymatvecmul P((int n_obj [OBJNUM_MAX ], plural float dX [3 ], plural float K_x [OBJNU
M_MAX ][3 ][2 ]));

/* convert.m */
void conv P((int n_obj [OBJNUM_MAX ], plural int len_mask , float X0 [OBJNUM_MAX ], float
Y0 [OBJNUM_MAX ], float Z0 [OBJNUM_MAX ], plural float T [3 ][3 ], plural float u , plural
 float v , plural float w , plural float *X , plural float *Y , plural float *Z ));
void back_conv P((plural int len_mask , plural float X , plural float Y , plural float Z ,
 float X0 [OBJNUM_MAX ], float Y0 [OBJNUM_MAX ], float Z0 [OBJNUM_MAX ], plural float T_co
nv [3 ][3 ], plural float *u , plural float *v , plural float *w ));
void short_conv P((int n_obj [OBJNUM_MAX ], plural float X , plural float Y , plural float
 Z , plural float alpha [OBJNUM_MAX /2 ], plural float beta [OBJNUM_MAX /2 ], plural int l
en_mask ));
void form_sin_cos P((plural int len_mask , plural float theta , plural float phi , plural
float psi , plural float T [3 ][3 ]));

/* data.m */
void data P((int len_obj [OBJNUM_MAX ], int max_indx , plural float *d_r , plural float *d
_i ));
void range_data P((int index [OBJNUM_MAX ], int len_obj [OBJNUM_MAX ], int max_indx , plur
al float *range_dat ));

/* GaussRand.m */
void GaussRand P((plural float *gauss1 ));
```

```
/* diff.m */
void diff P((int iter , int sel , int n_obj [OBJNUM_MAX ], int len_obj [OBJNUM_MAX ], plur
al float dat_r , plural float dat_i , plural float range_dat , plural float *X , plural fl
oat *Y , plural float *Z , float X0 [OBJNUM_MAX ], float Y0 [OBJNUM_MAX ], float Z0 [OBJNU
M_MAX ], plural float cov_out [OBJNUM_MAX ][3 ][2 ], plural float C_x [OBJNUM_MAX ][3 ][2
], plural int len_mask ));

/* like_calculation.m */
void like P((int option , int sel , int n_obj [OBJNUM_MAX ], plural float dat_r , plural f
loat dat_i , plural float range_dat , plural float alpha [OBJNUM_MAX /2 ], plural float be
ta [OBJNUM_MAX /2 ], float *likelihood , plural float X , plural float Y , plural float Z
));
void like_track_birth P((int M , plural float dat_r , plural float dat_i , plural float ra
nge_dat , float *likelihood , float X0 [OBJNUM_MAX ], float Y0 [OBJNUM_MAX ], float Z0 [OB
JNUM_MAX ]));

/* write_socket.m */
void write_socket P((int num_obj , int sock1 , int len_obj [OBJNUM_MAX ], int n_obj [OBJNU
M_MAX ], int len_track , int move_type , int accept_reject , plural float X , plural float
 Y , plural float Z , int index [OBJNUM_MAX ]));

/* initialization.m */
void initialization P((void ));

/* rotation.m */
void rotation P((int sel , int n_obj [OBJNUM ], int len_obj [OBJNUM ], float *p , float *q
 , float *r , float rot_mat [3 ][3 ]));

/* shiftwindow.m */
void shiftwindow P((int *shift_yes , int sel , int n_obj [OBJNUM_MAX ], plural float *dat_
r , plural float *dat_i , plural float *u , plural float *v , plural float *w , plural flo
at *X , plural float *Y , plural float *Z , float X0 [OBJNUM_MAX ], float Y0 [OBJNUM_MAX ]
, float Z0 [OBJNUM_MAX ], plural float *theta , plural float *phi , plural float *psi , pl
ural float *range_dat ));

/* init_jupiter.m */
void init_jupiter P((void ));

#undef P
```

```
#include <../include/params2.h>
```

```
#define BOXON 150
#define ORSTART 300      /* start orientation processing */
#define ORSKIP 5         /* number of steps to skip between orient disp */
#define NUM_SHUF 100 /* number of time shuffles is displayed */
#define OREND 1200       /* end orientation processing */
#define STOPF 1200  /* stop all */
```

```
#include "realparms.h"

/* round off routine */
#define round(r_x) ((int)f_floor((r_x) + 0.5))

/* define transup and transrt */
#define transup(u_data,u_up)             \
    ((u_up > 0) ? nowrapN(u_up,u_data): nowrapS((-1*u_up),u_data))
#define transrt(r_data,r_right)          \
    ((r_right > 0) ? nowrapE(r_right,r_data): nowrapW((-1*r_right),r_data))

/* define actual translate */
#define translate(t_data,t_right,t_up)       \
    (transup(transrt(t_data,t_right),t_up))

/* error routine */
#define enderror(n)              \
{ printf("error occured (%d).\nexiting program.\n",n); \
  exit(-1); }

/* pause routine */
#define pause(p_flag)                \
{   int p_temp;              \
        if (p_flag != 0) {           \
           sleep(1); \
           printf("\n\npause..\n\n");            \
           scanf("%c\n",&p_temp);        \
} }

/* define nowrap xnets */
#define nowrapN(d,val) ((iyproc < d) ? 0:xnetN[d].val)
#define nowrapS(d,val) ((iyproc >= nyproc-(d)) ? 0:xnetS[d].val)
#define nowrapW(d,val) ((ixproc < d) ? 0:xnetW[d].val)
#define nowrapE(d,val) ((ixproc >= nxproc-(d)) ? 0:xnetE[d].val)

int selectOne();
int close();
/* int exit(); */
int system();
```

```
/* fileio.h */
#define PACK 1024

/* general.h */
#define MAXVAL 32767
#define XSIZE 64
#define YSIZE 64
#define TRUE 1
#define FALSE 0
/* typedef int boolean; */

#define  NEVER 0
#define  ALWAYS 1

/* pause routine */
#define pause(p_flag)                    \
{   int p_temp;                  \
        if (p_flag != 0) {               \
           sleep(1); \
           printf("\n\npause..\n\n");              \
           scanf("%c\n",&p_temp);        \
} }

/* translate macro */
/* define nowrap xnets */
#define nowrapN(d,val) ((iyproc < d) ? 0:xnetN[d].val)
#define nowrapS(d,val) ((iyproc >= nyproc-(d)) ? 0:xnetS[d].val)
#define nowrapW(d,val) ((ixproc < d) ? 0:xnetW[d].val)
#define nowrapE(d,val) ((ixproc >= nxproc-(d)) ? 0:xnetE[d].val)

/* orient.h */
#define  TRIG_SIZE 51
#define  TRIG_MAX 2*M_PI
#define  TRIG_STEP (TRIG_MAX/TRIG_SIZE)

#define EOD 400

/* num.h */

#define NUM 5544
/* #define NUM 8000 */

/* picker.h */
#define PITCHES 19
#define DELTA 10
#define TWODELTA 20

#define STEP 18

int selectOne();
int read();
int write();
int close();
```

```
#define tracksize       1505
#define lambda  0.1
#define force_var_x     1.0
#define force_var_y     1.0
#define force_var_z     1.0
#define mean_diff       20
#define S_R     1.0
#define S_I     1.0
#define N_R     0.1
#define N_I     0.1
#define ran_var 3.0
#define OBJNUM 1
#define OBJNUM_TRU 1
#define OBJNUM_MAX 4
#define max(a,b)     ((a) > (b) ? (a) : (b))
#define min(a,b)     ((a) < (b) ? (a) : (b))
#define HighRes_var 1000.0
#define pitch_tmpl_stp 10
#define roll_tmpl_stp 10
#define image_read_stp 5

/*** for display file *****/
#define DIM 3
#define BSIZE 4500
#define RATIO 4

#define XSIZE 64
#define YSIZE 64
#define TRUE 1
#define FALSE 0

#define MAX_ITERATION 10000

#define SOCKET

/*  Jump move codes  */

#define TRACK_BIRTH 0
#define TRACK_DEATH 1
#define SEG_BIRTH 2
#define SEG_DEATH 3

#define BEAMWIDTH 20.0
```

```
#define X 0
#define Y 1
#define Z 2
#define PITCH 0
#define YAW 1
#define ROLL 2
#define DIM 3
#define XSIZE 64
#define YSIZE 64
#define OSIZE 10
#define UP 0
#define RATIO 5.0
#define EOD 400
#define INPORT 1991 /* input from anuj's program */
#define OUTPORT 1992    /* output to my MPP program */
#define IN2PORT 1993    /* input from my MPP program */

void swap_it();
void draw_templ();
int read_sock();
int enderror();
int getmove();
void grab_n_dump();
void shuffle();

void open_rp_win();
void def_rp();
void draw_rps();
```

The current directory is:

~atr/demo/marksrc/lib

This directory contains routines for performing target detection using simulated data from a cross array of radar sensors. Source code for the following routines appears in this directory:

GaussRand.m	generates Gaussian pseudo-random numbers
data.m	computes direction vectors for sensor array
data_fft.m	computes FFT of direction vectors
my_reduce.m	columnwise summation routine for mpp
randseed.m	seeds random number generator
range_detect.m	estimates range to detected target
t_birth.m	hypothesizes detection of a new target and tests observed data for support of a new target
t_death.m	hypothesizes removal of a detected target and tests observed data for support of one fewer target in the scene
xyz2azel.m	converts from rectangular to spherical coordinates

See the source code for further comments.

```
/*
        This routine computes a Gaussian random variable with
        zero mean and unit variance.  It is borrowed from
        Anuj.
*/


#include <stdio.h>
#include <mpl.h>
#include <math.h>
#include <su/values.h>
#include <su/stdlib.h>
#include "head.h"

void GaussRand(plural float *gauss1)
{
        plural float rand, theta, r;
        float pi,scale;
        plural float g,h;

        pi = 4.0*atan(1.0);
        scale=2147483647.0;

        theta=2*pi*(p_random()/scale);

        rand=p_random()/scale;
        while (rand==0)
            rand=p_random()/scale;
        r=fp_sqrt(-2.0*fp_log(rand));

        *gauss1=r*fp_cos(theta);
}
```

```
/* This routine computes the simulated direction vector for a target located
at elevation alpha, azimuth beta according to the array sensor model.  The
signal has real, imaginary amplitude s_r, s_i, and the added Gaussian noise
has amplitude n_r, n_i.  The vector is returned in the array d, with d[0],
d[1] being the real, imaginary parts, respectively, and the vector stored in
[0][0]...[63][0], repeated in every column.  */

#include<stdio.h>
#include<math.h>
#include<mpl.h>
#include<mpml.h>
#include "head.h"

extern plural float index_u, index_d;

void my_data(float alpha, float beta, float s_r,float s_i,float n_r,float n_i,
        plural float d[2])
{

        plural float rand_r,rand_i;
        plural float temp,dr,di;
        float pi,term;

        printf("Data:  alpha: %f beta: %f\n",alpha,beta);

        pi = 4.0*atan(1.0);

        alpha = alpha * pi / 180.0;
        beta = beta * pi / 180.0;

        temp = pi*(index_d* cos(alpha)*sin(beta)
                        + index_u * cos(alpha)*cos(beta));
        dr = fp_cos(temp);
        di = -fp_sin(temp);

        GaussRand(&rand_r);
        GaussRand(&rand_i);

        d[0] = (s_r*dr - s_i*di) + n_r*rand_r;
        d[1] = (s_r*di + s_i*dr) + n_i*rand_i;

}
```

```
/* This routine computes the FFT of the direction vector Data to
obtain the data likelihood.  The routines cfft2d_sizes, cfft2d_init
provide initialization for the MASPAR library cfft2d routine, and you
should consult the MASPAR library documentation for the meaning of
their parameters.

The input data vector is rearranged in the form of the array geometry
(a cross array) and stored in dt[].  Then the 2D FFT is performed,
with the results also in dt[], and the magnitude-squared of dt[] is
stored in mdt and returned.  */

#include <stdio.h>
#include <mpl.h>
#include <math.h>
#include <mpml.h>
#include <su/stdlib.h>
#include "head.h"

/***************  TRACK BIRTH   *******************/

plural float data_fft(plural float *Data)
{

        float pi;
        plural float dt[2], mdt;
        int i;

/*  FFT variables  */
        int zxs, zys, wss, wps = 0;
        float *ws;
        plural float *wp, *work, sumsq;

        pi = 4.0*atan(1.0);


        cfft2d_sizes(64,64,&zxs,&zys,&wss,&wps);
        ws = malloc(wss * sizeof(float) * 2);
        wp = p_malloc(wps * sizeof(float) * 2);
        work = p_malloc(sizeof(float) * 2);
        cfft2d_init(ws, wp, 64, 64);

/*  rearrange data vector for fft  */

        dt[0] = 0; dt[1] = 0;

        for(i=0;i<32;i++) {
                proc[32][i+16].dt[0]  = proc[i][0].Data[0];
                proc[32][i+16].dt[1]  = proc[i][0].Data[1];
                proc[i+16][32].dt[0]  = proc[i+32][0].Data[0];
                proc[i+16][32].dt[1]  = proc[i+32][0].Data[1];
        }

/* do it*/

        cfft2d(dt, ws, wp, 64, 64, work);

        mdt = dt[0] * dt[0] + dt[1] * dt[1];

        return mdt;
}
```

```
/* This routine computes the columnwise sum of a plural src and stores
the result in sum.  */

#include <mpl.h>
#include <math.h>
#include <stdio.h>
#include "head.h"

void my_reduce_y(register plural float src, plural float *sum)
{

        register plural int active;
        all active = 0;
        active = 1;

        all {
                register int d;

                if (!active) src = 0;
                for (d=2; d<=64; d = d<<1)
                        if ((iyproc & d-1) == d-1)
                                src += xnetpN[d/2].src;
        }
        *sum = src;
        return;
}

/* This routine computes the sum of the [0][0]...[63][63] elements of
the plural src, and stores the result in sum. */

void my_reduce(register plural float src, float *res1)
{
        register plural int active;

        all active = 0;
        active = 1;

        all {
                register int d;

                if (!active) src = 0;
                for (d=2; d<=128/2; d = d<<1)
                        if ((ixproc & d-1) == d-1)
                                src += xnetpW[d/2].src;
                for (d=2; d<=64; d = d<<1)
                        if ((iyproc & d-1) == d-1)
                                src += xnetpN[d/2].src;
        }
        *res1 = proc[63][63].src;
        return;
}

/* This routine converts the electrical angles row, col into an
elevation/azimuth pair a,b, in radians.  quadrant is used to compute
the geometric quadrant of the electrical angles and the resulting
el/az are transformed accordingly. */

void azelconv(int row, int col, float *a, float *b)
{
        float pi;
        float phi1, phi2;
        int quadrant = 0;

        pi = 4.0*atan(1.0);

        if (row > 32) {
                row = 64 - row;
                quadrant += 1;
        }

        if (col > 32) {
                col = 64 - col;
                quadrant += 2;
        }

        printf("row: %d col: %d\n",row,col);

        phi1 = 2 * col / 64.0;
        phi2 = 2 * row / 64.0;


        if (phi1*phi1 + phi2*phi2 >= 1) *a = 0;
        else *a = acos(sqrt(phi1*phi1 + phi2*phi2));

        if (phi1 == 0) *b = pi / 2;
        else *b = atan(phi2/phi1);

        if (quadrant == 1) *b = pi - *b;
        else if (quadrant == 0) *b = pi + *b;
        else if (quadrant == 2) *b = 2*pi - *b;

        *a = *a * 180 / pi;
        *b = *b * 180 / pi;

        return;

}
```

```
/* This routine is used to generate a random seed for the random
number generator. */

#include <sys/time.h>

long get_rand_seed()
{
        struct timeval tp;
        struct timezone tzp;

        gettimeofday(&tp,&tzp);
        return tp.tv_usec;

}
```

```
/* This routine performs range detection.  It is assumed that a
hypothesized target has been located at el, az.  It then compares the
actual target location with the hypothesis and if the actual location
is greater than BEAMWIDTH-squared away then range detection fails and
a negative range is returned.  Otherwise the actual range is returned;
range detection in our model is somewhat trivial at this stage. */

#include<stdio.h>
#include<math.h>
#include "head.h"
#include "realparms.h"

extern float track[OBJNUM_MAX][tracksize][3],pitch[OBJNUM_MAX][tracksize],
             yaw[OBJNUM_MAX][tracksize], roll[OBJNUM_MAX][tracksize];

float range_detect(float el, float az, int index[], int M)
{
        float range, config[2];
        int r, found;

        found = -1;
        for(r=0;r<OBJNUM_TRU;r++) {
                xyz2azel(track[r][0], config);
                if ((el-config[0])*(el-config[0])+(az-config[1])*
                        (az-config[1]) < BEAMWIDTH*BEAMWIDTH) found = r;
        }
        if (found == -1) range = -1;
        else {
                range = sqrt(track[found][0][0]*track[found][0][0] +
                             track[found][0][1]*track[found][0][1] +
                             track[found][0][2]*track[found][0][2]);
                index[M] = found;
        }
        printf("range_detect: found %d range %.2f\n",found,range);
        return range;

}
```

```
/* This is the main routine of the detection library.  It takes
simulated data Data and attempts to detect a new target given this
data. pos[][3] is an array of x,y,z positions of targets that have
already been detected.  M is the number of targets that have already
been detected.  rej_mask is a plural which has a 0 for electrical
angles where hypothesized targets have already been rejected, and 1
elsewhere.

First, it removes the data from already-detected targets to obtain the
residual data.  It then calls data_fft to compute the 2D FFT of that
data.  The result is multiplied by rej_mask to remove spurious peaks
and then the result is converted from log-likelihoods to likelihoods.
Finally histogram (Gibbs) selection is performed on the
log-likelihoods and the selected peak is returned in result with the
azimuth, elevation in [0][0], [0][1], and the electrical angles in
[1][0], [1][1] (to update rej_mask in the main routine).  */


#include <stdio.h>
#include <mpl.h>
#include <math.h>
#include <stdlib.h>
/*#include <mpl/math.h>*/
#include <su/stdlib.h>
#include <mpml.h>
#include <values.h>
#include "head.h"
#include "realparms.h"


extern plural float index_u, index_d;

/*********************** TRACK BIRTH *********************/

plural float t_birth(plural float *Data, float pos[][3], int M, plural int rej_mask)
{
        plural float tb, tblike;
        plural float t[2];
        plural float result = 0;
        float maxsum, norm;
        float angle[2];
        float rv, az, el;
        int i,j;

/**************  JUMP MOVES    *********************/

        for(i=0;i<M;i++) {
                xyz2azel(pos[i],angle);
                my_data(angle[0], angle[1], 1, 1, 0, 0, t);
                Data[0] = Data[0] - t[0];
                Data[1] = Data[1] - t[1];
        }

        tb = data_fft(Data);

/* Remove previously rejected peaks */
        tb = tb * rej_mask;

/***************  LIKLIEHOODS  **********************/
/*  convert to likliehoods  */
        maxsum = reduceMaxf(tb);
        tblike = 0;
        if ((ixproc < 64) && (iyproc < 64)) {
                tblike = fp_exp(tb - maxsum);
        }
        my_reduce(tblike, &norm);
        printf("maxsum: %.4f norm: %.4f\n",maxsum,norm);
        for(i=0;i<64;i++) {
                for(j=0;j<64;j++) {
/*                      printf("%d %d %.4f %.4f\n",i,j,proc[i][j].tb,proc[i][j].tblike); *
/
                }
        }
        tblike = tblike / norm;


/*  Histogram selection  */
        srandom((int) get_rand_seed());
        rv = (random()*1.0)/MAXINT;
        i = 0; j = 0;
        while (rv >= 0) {
/*              printf("i: %d j: %d rv: %.2f like: %.2f\n",i,j,rv,proc[i][j].tblike); */
                rv = rv - proc[i][j].tblike;
                j++;
                if (j > 63) { j = 0; i++; }
        }

        if (j == 0) { j = 63; i--; }
        else { j--; }

/*  Convert the result */

        printf("Max frequency:  %d %d \n",i,j);

        azelconv(i, j, &az, &el);
        proc[0].result = az;
        proc[1].result = el;
        proc[1][0].result = i;
        proc[1][1].result = j;

/*  print results  */

        printf("Track Birth\n");

        printf("Chose:  %f  %f  Liklihood:  %f \n",az,el,proc[i][j].tblike);

        return result;

}
```

```
#include <stdio.h>
#include <mpl.h>
#include "head.h"

/********************  TRACK DEATH  ******************/

plural float t_death(plural float *Data, float config[][2], int M)
{

        plural float td = 0, dt[2], t[2], sumsq, sum;

        int i,j;

        for(i=0;i<M;i++) {
          dt[0] = 0;
          dt[1] = 0;
          for(j=0;j<M;j++) {
            if (!(i == j)) {
              my_data(config[j][0],config[j][1],1,0,0,0,t);
              dt[0] = dt[0] + t[0]; dt[1] = dt[1] + t[1];
            }
          }
          sumsq = (dt[0] - Data[0]) * (dt[0] - Data[0]) +
                (dt[1] - Data[1]) * (dt[1] - Data[1]);
          my_reduce_y(-sumsq, &sum);
          proc[0][i].td = proc[63][0].sum;
        }

        return td;

}
```

```
/* This routine converts an x,y,z position into an azimuth, elevation
pair in the array config. */

#include <stdio.h>
#include <math.h>

void xyz2azel(float pos[3], float config[2])
{
        float pi;

/*      printf("xyz2azel: in: %.2f %.2f %.2f\n",pos[0],pos[1],pos[2]);*/

        if (pos[0] == 0.0 && pos[1] == 0.0) {
                config[0] = 0; config[1] = 0;
        } else {

        pi = atan(1.0) * 4;
        config[0]=asin(pos[2]/sqrt(pos[0]*pos[0]+pos[1]*pos[1]+pos[2]*pos[2]));
        config[1]=asin(pos[1]/sqrt(pos[0]*pos[0]+pos[1]*pos[1]));
        printf("config[1]: %.2f\n",config[1]);
        if (pos[0] < 0) config[1] = pi - config[1];
        if (pos[1] < 0 && pos[0] > 0) config[1] += 2 * pi;

        config[0] = config[0] * 180 / pi;
        config[1] = config[1] * 180 / pi;

        }

/*      printf("xyz2azel: out: %.2f %.2f\n",config[0],config[1]);*/


}
```

The current directory is:

~atr/demo/sgi

This directory contains the C program

display_rps.c

Which is responsible for displaying the results of orientation estimation using simulated range profile data. The program receives orientations and range profile data from the mpp program via a socket connection, uses routines from the standard SGI graphics library to draw range profiles and uses the routines in the directory:

~atr/demo/sgi/lib

to do the 3-D rendering of the aircraft at various orientations. See the source code for further comments.

```
/************************************************************************
*                                                                       *
*       display_rps                                                     *
*                                                                       *
*************************************************************************
*                                                                       *
* written by: Mohammad Faisal and Steve Jacobs                          *
* comments: Steve Jacobs                                                *
*                                                                       *
*************************************************************************
*                                                                       *
* This program is responsible for visulaization of the results of our   *
* joint tracking and orientation estimation algorithm.  In particular,  *
* both true and estimated orientations are provided by the estimation   *
* algorithm to this display program via a socket connection.  This      *
* program reads the values from the socket, draws a 3-D rendering of    *
* the target at the various orientations and draws colored traces to    *
* represent the range profiles.  The nature of socket communication     *
* requires a high degree of coordination between estimation algorithm   *
* and this program.  What follows is a brief outline of the program     *
* structure.                                                            *
*                                                                       *
*       open windows, initialize graphics                               *
*       open socket                                                     *
*       for each invocation of orientation estimation                   *
*               read true orientation                                   *
*               display target at true orientation                      *
*               read observed range profile                             *
*               display observed range profile                          *
*               for each orientation estimate                           *
*                       read estimated orientation                      *
*                       display target at estimated orientation         *
*               for final estimate only                                 *
*                       read estimated range profile                    *
*                       display estimated range profile                 *
*                                                                       *
************************************************************************/

#include <stdio.h>
#include <math.h>
#include <sys/stat.h>
#include <strings.h>

/* SGI graphics libraries */

#include <gl/gl.h>
#include <gl/device.h>

/* The header file flip.h and the C code in flip.c contain the
   routines flip_read and flip_draw.  These routines are used for
   reading data and performing the display, respectively, of a 3-D
   rendering of the target. */

#include <flip.h>

/* font manager libraries */

#include <fcntl.h>
#include <fmclient.h>
```

```
/* The parameter NUM_DIFF defines the fixed number of iterations
   of the orientation estimation procedure which will be performed
   each time it is called.  The value set here must be equal to that
   defined in the mpp software which perfroms orientation estimation
   so that this display program will know how many orientation
   estimates will accompany each true orientation. */

#define NUM_DIFF 4

long estwin,truewin,flipwin, true_rp_win, est_rp_win;
Object axis;
float vertnew[2];

/* The following declarations and initializations are necessary for
   displaying a 3-D rendering of the target of interest.  These lines
   were copied from a similar program written by rst. */

/* global structures for lighting parameters */
Matrix Identity = { 1, 0, 0, 0,  0, 1, 0, 0,  0, 0, 1, 0,  0, 0, 0, 1 };

/* define material */

static float shufmat[] = {
        AMBIENT, 0.2, 0.2, 0.2,
        DIFFUSE, 0.2, 0.5, 0.7,
        SPECULAR, 0.5, 0.5, 0.5,
        SHININESS, 65,
        LMNULL,
};

static float lt_front[] = {
     LCOLOR, 1., 1., 1.,
     POSITION, 0., 0., 1., 0.,
     LMNULL
};

static float mat[] = {
        AMBIENT, 0., 0., 0.,
        DIFFUSE, 0.5, 0.5, 0.5,
        SPECULAR, 1, 1, 1,
        SHININESS, 64,
        LMNULL,
};

/* Because dec and SGI have different standards for the representation
   of integer and floats, any data which is communicated from the mpp
   to an SGI machine must undergo byte swapping in order to retain
   its intended numerical value.  The following routine for performing
   byte swapping was copied from another file.  The routine was written
   by rst. */

void swap_it(data)
unsigned char *data;
{
        unsigned char cptr[2];

        cptr[0] = data[0];
        cptr[1] = data[1];

        data[0] = data[3];
        data[1] = data[2];
        data[2] = cptr[1];
        data[3] = cptr[0];

}

/* The following routine, also written by rst, is responsible for
   reading data from a communication socket which has already been
   established. */

int read_sock(sock,ptr,sz)
int sock,sz;
char *ptr;
{
        int rd=0;
        int ret;

        while (rd < sz) {
                ret = read(sock,ptr,sz-rd);
                if (ret <= 0) return(ret);
                ptr += ret;
                rd += ret;
        }
        return(rd);
}

main()
{

 int i,j,k,l,diff_count;         /* generic loop indices */
 int sock;                       /* id for communication socket */
 float pitch, yaw, roll;         /* orientation angles */
 float est_pitch, est_yaw, est_roll;
                                 /* estimates of orientation angles */
 char fname1[20];                /* file name for 3-D target model */
 char name[80];                  /* storage for window names */
 unsigned char test_flag;        /* flag for checking socket performance */
 flip_type ft_X29;               /* identifier for 3-D rendering of target */
 float recv_temp;                /* storage for data received from socket */
 float true_rp[1024];            /* observed range profile */
 float est_rp[1024];             /* rp from library at estimated orienataion */
 Object line_a, line_b;          /* graphics objects for drawing rp's */
 float true_max;                 /* maximum value of true range profile */
 fmfonthandle font1, font15;     /* handles for times-roman font */
 int dev;                        /* storage for ESC key window exiting */
 short val;                      /* storage for ESC key window exiting */

        /*printf("JUPITER: show_rps is running\n");
        fflush(stdout);*/

/* Initialize fontmanager, set up fonts */

        fminit();
        if ((font1=fmfindfont("Times-Roman")) == 0) exit(1);
        font15 = fmscalefont(font1,15.0);
        fmsetfont(font15);

/* Initialize graphics. */

        if (dglopen("jupiter:0",DGLLOCAL) < 0) {
            printf("dgl open error\n");
            exit();
        }

/* Read data for 3-D rendering of target */

        sprintf(fname1,"plane.bin\0");
        flip_read(fname1,&ft_X29);

/* Open all graphics windows */

        window_opening();

/* Create graphics object for x-axis in range profile windows */

        makeaxis();

/* Establish id's for graphics objects */

        axis = 1; line_a = 2; line_b = 3;

/* The routine recvinit_p will accept wait for an attempt by the
   mpp to establish a communication socket, and establish it once
   the attempt is made.  This command must be executed before the
   mpp executes its sendinit_jupiter command. recvinit_p is included
   with this program by virtue of the -lsock option in the command
   line which is used to compile this program. */

        printf("JUPITER: ready to open socket\n");
        fflush(stdout);
        sock = recvinit_p(1989);
        printf("JUPITER: socket has been opened\n");
        fflush(stdout);

/* What follows is the main loop of the program.  True and estimated
   orientations and associated range profiles are read from the
   socket and displayed in appropriate windows */

        l = 0;
        while (!(qtest() && (dev=qread(&val)) == ESCKEY && val == 0)) {
                l++;
                winset(truewin);
                reshapeviewport();
                winset(true_rp_win);
                reshapeviewport();
                winset(flipwin);
                reshapeviewport();

        /*printf("Waiting for first socket communication\n");*/

/* Wait for arrival of test_flag before proceeding with socket
   communications */

        while(recvmine(sock,&test_flag,sizeof(test_flag)) != 1);
                /*printf("test_flag = %d\n",test_flag);*/
          if(test_flag == 5){

                /* Read true orientation from socket */

                read_sock(sock, &pitch,4);
                swap_it(&pitch);
                read_sock(sock, &yaw,4);
                swap_it(&yaw);
                read_sock(sock, &roll,4);
                swap_it(&roll);
                /*printf("jup actual: pitch = %f yaw = %f roll = %f\n",pitch, yaw, roll);*
/
                printf("%f \t %f \t %f \n", pitch, yaw, roll);
                fflush(stdout);
                        }

        true_max = 0.0;

/* Wait for arrival of test_flag before proceeding with socket
   communications */

        while(recvmine(sock,&test_flag,sizeof(test_flag)) != 1);

/* Read true range profile from socket */

        for (i=0; i<1024; i++)
        {
                read_sock(sock, true_rp+i, 4);
                swap_it(true_rp+i);

                /* retain maximum value of true range profile */

                if (true_rp[i] > true_max)
                        true_max = true_rp[i];
        }
        /*printf("True rp max value = %f \n",true_max);*/

/* Cretae graphics object for range profile */

                makeobj(line_a);
                bgnline();
                for (j=0;j<1024;j++)
                {
                        vertnew[0] = j;
                        vertnew[1] = true_rp[j];
                        v2f(vertnew);
                }
                endline();
                closeobj();

/* Draw axis, true range profile. */

                winset(true_rp_win);
                ortho2(0,1024,-true_max/20,true_max);
                pushmatrix();
                color(BLACK);
                clear();
                color(BLUE);
                linewidth(1);
                callobj(axis);
                color(RED);
                linewidth(1);
                callobj(line_a);
                popmatrix();
                swapbuffers();
```

```
/* The following loop displays orientation estimates as read from
   the sockets.  As the estimation algorithm is executed NUM_DIFF
   times, there are NUM_DIFF + 2 orientation estimates, provided
   in the following order:
                Initial guess
                Nearest grid point
                NUM_DIFF estimates.
   For all except the last estimate, only the orientation is
   sent accross the socket.  For the last estimate, both the
   orientation and the corresponding range profile are sent
   and displayed. */

        for (diff_count=0;diff_count<(NUM_DIFF+2);diff_count++) {

/* Wait for arrival of test_flag before proceeding with socket
   communications */

                while(recvmine(sock,&test_flag,sizeof(test_flag)) != 1);

                /* Read orientation guess from socket. */

                read_sock(sock, &est_pitch,4);
                swap_it(&est_pitch);
                read_sock(sock, &est_yaw,4);
                swap_it(&est_yaw);
                read_sock(sock, &est_roll,4);
                swap_it(&est_roll);
                /*printf("jup guess: pitch = %f yaw = %f roll = %f\n",est_pitch, est_yaw,
est_roll);*/
                fflush(stdout);


/* Draw 3-D rendering of target at true orientation. */

                winset(truewin);
                czclear(0x000000, 0x7fffff);
                pushmatrix();
                pushmatrix();
                RGBcolor(0,255,255);
                cmovi(-4,5,0);
                sprintf(name,"TRUE");
                fmprstr(name);
                popmatrix();
                rot(-90,'y');
                rot(-90,'x');
                rot(0,'z');
                rot(180-yaw, 'z');
                rot(pitch, 'x');
                rot(-roll, 'y');
                scale(15,15,15);
                flip_draw(ft_X29);
                popmatrix();
                swapbuffers();

/* Draw 3-D rendering of target at guessed orientation. */

                winset(flipwin);
                czclear(0x000000, 0x7fffff);
                pushmatrix();
                pushmatrix();
                RGBcolor(0,255,255);
                cmovi(-4,5,0);
                sprintf(name,"ESTIMATE");
                fmprstr(name);
                popmatrix();
                rot(-90,'y');
                rot(-90,'x');
                rot(0,'z');
                rot(180-est_yaw, 'z');
                rot(est_pitch, 'x');
                rot(-est_roll, 'y');
                scale(15,15,15);
                RGBcolor(0,255,255);
                flip_draw(ft_X29);
                popmatrix();
                swapbuffers();

                /*printf("Iteration number = %d\n",l);*/

/* If this is the last estimate, print out estimated angles*/

                if (diff_count == NUM_DIFF + 1)
                        printf("%f \t %f \t %f \n", est_pitch, est_yaw, est_roll);

/* If this is the last estimate, read estimated range profile and
   display. */

                if (diff_count == NUM_DIFF + 1) {
                while(recvmine(sock,&test_flag,sizeof(test_flag)) != 1);

/* Read estimated range profile from socket */

                for (i=0; i<1024; i++){
                        read_sock(sock, est_rp+i, 4);
                        swap_it(est_rp+i);
                }

/* Create graphics object for estimated range profile */

                        makeobj(line_b);
                        bgnline();
                        for (j=0;j<1024;j++)
                        {
                                vertnew[0] = j;
                                vertnew[1] = est_rp[j];
                                v2f(vertnew);
                        }
                        endline();
                        closeobj();

/* Draw axis, estimated rp. */

                        winset(est_rp_win);
                        ortho2(0,1024,-true_max/20,true_max);
                        color(BLACK);
                        clear();
                        color(BLUE);
                        linewidth(1);
                        callobj(axis);
                        color(YELLOW);
                        linewidth(1);
                        callobj(line_b);
                        swapbuffers();

                }

/* If this is the next-to-last estimate, clear the window for
   estimated range profile. */

                else if (diff_count == NUM_DIFF) {
                        /*printf("Clearing est_rp_win \n");*/
                        winset(est_rp_win);
                        color(BLACK);
                        clear();
                }

        }


 }

/* Program complete.  Close socket. */

        /*printf("JUPITER: ready to close socket\n");*/
        fflush(stdout);
        close(sock);
        /*printf("JUPITER: socket closed\n");*/
        fflush(stdout);
        gexit();

}


void window_opening()
{

/* Open window for 3-D target display. Initialize graphics parameters
   for this window. */

        noborder();
        prefposition(0,200,300,500);
        foreground();
        truewin = winopen("truth");
        color(BLACK);
        clear();
        qdevice(ESCKEY);
        RGBmode();
        doublebuffer();
        gconfig();
        lsetdepth(0, 0x7fffff);
        zbuffer(TRUE);
        mmode(MVIEWING);
        loadmatrix(Identity);
        ortho(-10,10,-10,10,-2*10,2*10);
        czclear(0x000000, 0x7fffff);
/* material and lighting definitions */
        lmdef(DEFMATERIAL, 1, 0, mat);
        lmdef(DEFLIGHT, 1, 0, lt_front);
        lmdef(DEFLMODEL, 1, 0, NULL);
```

```
/* material and lighting bindings */
       lmbind(MATERIAL, 1);
       lmbind(LMODEL, 1);
       lmbind(LIGHT0, 1);
       translate(0, 0, -10);


/* Open window for 3-D target display. Initialize graphics parameters
   for this window. */

       noborder();
       prefposition(200,400,300,500);
       foreground();
       flipwin = winopen("plane");
       color(BLACK);
       clear();
       qdevice(ESCKEY);
       RGBmode();
       doublebuffer();
       gconfig();
       lsetdepth(0, 0x7fffff);
       zbuffer(TRUE);
       mmode(MVIEWING);
       loadmatrix(Identity);
       ortho(-10,10,-10,10,-2*10,2*10);
       czclear(0x000000, 0x7fffff);
/* material and lighting definitions */
       lmdef(DEFMATERIAL, 1, 0, mat);
       lmdef(DEFLIGHT, 1, 0, lt_front);
       lmdef(DEFLMODEL, 1, 0, NULL);
/* material and lighting bindings */
       lmbind(MATERIAL, 1);
       lmbind(LMODEL, 1);
       lmbind(LIGHT0, 1);
       translate(0, 0, -10);

/* Open window for true rp display. */

       noborder();
       prefposition(0,200,100,300);
       foreground();
       true_rp_win = winopen("true range profile");
       color(BLACK);
       clear();
       qdevice(ESCKEY);

/* Open window for estimated rp display. */

       noborder();
       prefposition(200,400,100,300);
       foreground();
       est_rp_win = winopen("estimated range profile");
       color(BLACK);
       clear();
       qdevice(ESCKEY);
}

void makeaxis()
{
```

```
/* The object 'axis' is simply a line across the bottom of the
   rp display window */

        makeobj(axis);
        bgnline();
                vertnew[0] = 0;
                vertnew[1] = 0;
                v2f(vertnew);
                vertnew[0] = 1023;
                vertnew[1] = 0;
                v2f(vertnew);
        endline();
        closeobj();
}
```

Appendix A

Table of Contents

Automated Taget Tracking and Recognition Using Jump-diffusion Processes, by Anuj Srivastava, M.S. thesis, Washington University Depoartment of Electrical Engineering, December 1993. . . . . pg A-2

A. Srivastava, M.I. Miller, U. Grenander, "Multiple Target Direction of Arrival Tracking." IEEE Transactions on Signal Processing, Vol. 43, No. 5, May 1995, pp. 1282-1285. pg A-85

M.I.Miller, A. Srivastava, and U. Grenander, "Conditional Mean Estimation via Jump-Diffusion Processes in Multiple Target Tracking/Recognitjion," to appear in IEEE Transactions on Signal Processing, 1995. . . . . pg A-85

Short title: **Automated Tracking-Recognition** Srivastava, M. Sc. 1993

WASHINGTON UNIVERSITY
SEVER INSTITUTE OF TECHNOLOGY

Automated Target Tracking and Recognition Using Jump-Diffusion Processes

by

Anuj Srivastava

Prepared under the direction of Dr. M. I. Miller

A Thesis presented to the Sever Institute of
Washington University in partial fulfillment
of the requirements for the degree of

Master of Science

December, 1993

Saint Louis, Missouri

WASHINGTON UNIVERSITY
SEVER INSTITUTE OF TECHNOLOGY

ABSTRACT

Automated Target Tracking and Recognition Using Jump-Diffusion Processes

by Anuj Srivastava

ADVISOR : Dr. M. I. Miller

December, 1993
Saint Louis, Missouri

A new random sampling algorithm for recognition and tracking of an unknown number of targets and target types is presented. Taking a Bayesian approach we define a posterior measure on the parameter space by combining the observed data likelihood with a prior based on airplane dynamics. The Newtonian force equations governing the airplane motion are utilized to form the prior density on the airplane positions. The sampling algorithm based on *Jump-Diffusion* processes, first introduced by [Grenander,Miller] (1991), is derived for generating high probability estimates of target positions, orientations and types from the posterior measure. Results are presented from its joint implementation on the Silicon Graphics workstation and the DECmpp SIMD machine distributing data-simulation, visualization and computation over the network.

TABLE OF CONTENTS

1 **Introduction** A-10

Target Tracking and Recognition A-10

Bayesian Posterior A-12

Random Sampling Approach A-14

Results and Contribution A-15

2 **Scene Parameterization** A-17

Parameter Set A-17

Parametric Space A-19

3 **Bayesian Posterior** A-21

Prior Density on Parameter Space $\mathcal{X}_t$ A-22

Analysis of Airplane Motion A-22

Prior on Rotational Motion A-26

Data Likelihood A-26

Tracking A-26

Imaging A-28

Bayesian Posterior A-32

4 **Estimation Through Random Sampling** A-34

Random Sampling A-34

Why Random Sampling A-34
Jump-Diffusion Sampling Algorithm A-35
Jump Process A-37
Diffusion Process A-41
Ergodic Result A-42

5 Implementation and Results A-44
Data Simulation A-45
Estimation Algorithm A-45
Parallel Processing A-48
Remote Visualization A-49

6 Tracking the Direction of Arrivals of Multiple Moving Targets A-51
Bayesian Posterior A-52
Data Likelihood A-52
von-Mises Prior A-53
Sampling the Multi-Track Space A-54
Diffusion S.D.E.'s A-54
Jump process A-55
Ergodic Theorem A-57
Implementation and Results A-58

7 Future Work A-61

8 ACKNOWLEDGMENTS A-63

9 Appendix A-64

Cross-Array Manifold A-65

Airplane Motion Based on Rigid Body Dynamics A-67

Jump and Diffusion Processes A-69

Jump Processes A-69

Diffusion Processes A-70

Proof of Theorem 1 . . . . A-72

Proof of Corollary 1 . . . . A-75

Proof of Theorem 2 . . . . A-76

10 GLOSSARY A-78

11 Bibliography A-80

12 Vita A-84

LIST OF FIGURES

1.1 An airplane target being observed by a uniform cross-array of sensors. A-12

2.1 3-D target generator $g \in \mathcal{G}^o$ at the origin (left panel) and after applying a rotation transformation (right panel). A-18

3.1 The observed target located at position $\vec{p}(s)$, oriented at $\vec{\phi}(s)$ with body frame velocities $\vec{v}(s)$. A-22

3.2 The figure displays the cross array of isotropic sensors at half wavelength spacing, which observes the angular location of the target. . . A-27

3.3 The projection transformation $\mathcal{P}$ converting a 3D volume into a 2D image on a discrete lattice $\mathcal{L}$ for a single target. A-30

3.4 The figure shows the target projected onto a 64×64 2-D lattice with additive noise at four different time instants. A-31

3.5 The figure shows the azimuth-elevation spatial power spectrum of the narrowband tracking data generated via MVDR method at four different instants of time (bright is low power, dark is high power). . . A-32

5.1 Jump-diffusion estimation of a portion of the track. The true track is drawn in grey while the estimates overlap in white. The estimation proceeds via a sequence of jump moves with the diffusion cycles performed between moves. A-48

5.2 3D track estimation: The left panel shows the actual track drawn in gray with the mesh representing ground supporting the observation system in the inertial frame of reference. The right panel displays the results from the single track estimation with the estimates drawn in white. A-49

6.1 The linear array of uniformly placed isotropic sensors to track the direction of arrival of multiple signal sources. A-51

6.2 The two kinds of jump moves: addition of a track and addition of a track-segment at the end of existing track estimate. The actual tracks are shown in grey with the estimates overlapping in white. Underlying spatial power spectrum generated via MVDR method shows the noisy data. A-59

6.3 Estimation of a four-track scene: the upper left panel shows the true tracks in grey with the spatial power spectrum generated via MVDR from the noisy data. The other three panels show the successive stages of the algorithm with the final result shown in the lower right panel. A-60

9.1 The figure shows the indexed sequence of sensor elements arranged in cross-array geometry. A-65

Automated Target Tracking and Recognition Using Jump-Diffusion Processes

1. Introduction

1.1 Target Tracking and Recognition

Automated target tracking and recognition are well known problems in the signal processing and control system's literature. They belong to a class of problems which utilize time records of multiple sensors to estimate the charaterstics of arriving signals. The goal is to identify and track the motion of multiple signal sources in the observation space using the data collected from various observation platforms. There has been a great deal of published work in the control literature on multiple target tracking posed as a state estimation problem [1, 2, 3, 4]. Much of it involves the use of Kalman filter based techniques. It has been realized that the observed data are non-linear in target parameters thereby emphasizing the use of the extended Kalman filters. This represents a set of linear approximations valid in some particular scenarios, but the general non-linear problem remains to be completely solved. Recently, there has been a considerable interest in tracking the directions of arriving signals from multiple moving sources recorded by an array of passive sensors [5, 6, 7, 8, 9]. In these non-linear data models, several variations of the gradient based techniques are used to solve the problem in mostly maximum-likelihood settings. But most of the workers utilize certain simplifying assumptions which are not always valid in a general tracking scenario. For example, targets may be assumed stationary between sample times with multiple (~ 100) snapshots at each sample time whereas, in general, for a moving target, each data sample reflects a new position. Also, though even some researchers mention the use of target dynamics for tracking, it is yet to be adequately utilized. This is partly due to the simplifying assumptions like restricted motion and partly due to the efforts to solve the tracking and recognition problems separately. In part, this is one major result of this thesis.

In the work presented here we define a random sampling based solution for the tracking and recognition problems in a general setting. More precisely, the problem is defined as follows: there are an unknown number of non-cooperative signal sources in the observation space of the sensor-system. The aim is to identify them and track their motion or, in other words, estimate their positions, orientations and types using the observed data. The observations are collected from multiple sensors; a narrowband sensor array providing azimuth-elevation data for object tracking (Figure 1.1), a range radar providing the target range data, or various wideband radars providing detailed information about the target-type and orientation. Also, for moving targets, we model a data sample for each target position. As in [10], we utilize the results from recognition component of the algorithm to improve upon the tracking results. In fact, we observe that the tracking and recognition should be treated as one unified problem as only the joint solution is optimal. The rotational and

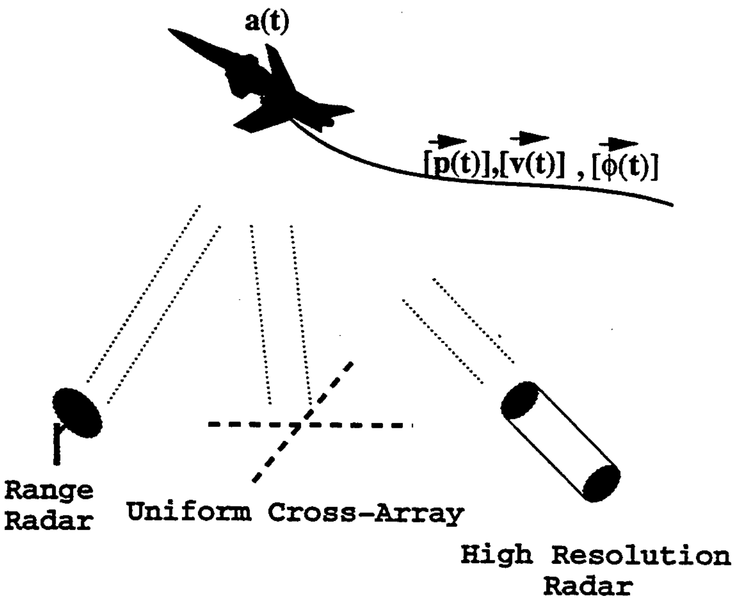


Figure 1.1: An airplane target being observed by a uniform cross-array of sensors.

translational motions of the airplane are coupled to each other by a set of differential equations modeling the motion. It results in a dynamics-based prior, on the airplane-positions, parameterized by the rotational parameters, thereby connecting the tracking and recognition components together.

1.2 Bayesian Posterior

We use a Bayesian approach by defining a posterior probability on the space of target parameters conditioned on the set of observed data. Notice that the posterior distribution $\pi_t(\cdot)$ changes with time as the data set is incremented at each observation time t. The parameter set describing a multiple target scene includes the positions, orientations and target-type associated with each target for the duration of its stay in the scene. The targets appear and disappear from the scene at random times so their dwell times are also to be estimated by the algorithm. The parameter space is defined to be $\mathcal{X}_t = \bigcup_{M=0}^{\infty} \mathcal{X}_t(M)$, a countably infinite union of subspaces, each corresponding to a specific value of M, the number of targets. The search for estimates is performed across the subspaces $\mathcal{X}_t(M)$ signifying the simultaneous model order estimation.

We utilize the collected data up to any fixed time t to generate minimum-mean-square-error (*MMSE*) estimates of the parameters describing target paths from initial time t_0 to t. It is well known that the *MMSE* estimate is the conditional mean of the parameter under the posterior distribution. Therefore, we perform a random sampling from the posterior distribution at time t based on Jump-Diffusion processes to estimate these conditional means [11, 12, 13].

We assume that the underlying scene consists of a set G of generators $g \in \mathcal{G}$ which can be observed by an ideal (with no loss of information) observer and are completely parameterized by a parameter set. The actual observer, however, may only be able to see the elements with loss of information due to projection, observation noise and/or limited accuracy in the sensor. Denote the operation which transforms the ideal G into some data set $I^{\mathcal{D}}$, by $\mathcal{D} : \mathcal{G} \rightarrow \mathcal{I}^{\mathcal{D}}$, where $\mathcal{D}$ is the deformation mechanism, both random and deterministic.

Highly probable candidate scenes are defined via a posterior probability of the parameter vector $\vec{x}$ representing ideal G given the measured data up to time t, $I_t^{\mathcal{D}}$ by

$$\pi_t(\vec{x}) = \frac{1}{Z} e^{-E_t(\vec{x})} = \frac{1}{Z} e^{-(P_t(\vec{x}) + L_t(\vec{x}))} \quad ,$$

where $L_t(\vec{x})$ is the potential associated with the likelihood of the data $I^{\mathcal{D}}(t)$ and $P_t(\vec{x})$ is the potential associated with the prior distribution on the parameter space $\mathcal{X}_t$.

In the problem solved here, the data $I_t^{\mathcal{D}}$ has multiple components corresponding to the various sensors:

$$I_t^{\mathcal{D}} \equiv \left(I_t^{\mathcal{D}}(1), I_t^{\mathcal{D}}(2), \ldots\right) .$$

We only include two sensor systems, one for tracking and one for high-resolution imaging. For tracking, we assume a narrowband cross array sensitive to the elevation and azimuth locations of the target, and a range radar measuring target range locations. The likelihood follows from a standard narrowband signal model developed in [14] and used in [15, 16, 17, 18, 19]. The imaging data obtained from an optical sensing radar is modeled as the 2-D orthographic projection of the true 3-D object under far field assumption.

The prior on airplane motion is based on the dynamics of target motion and follows that described in Srivastava et al. [17] in which the force equations governing the motion of targets are utilized to form a prior density on the track parameter space. This prior combined with the data likelihood forms the required posterior distribution.

1.3 Random Sampling Approach

Our approach is to construct a jump-diffusion Markov process, following that outlined in Grenander and Miller [11], having the limiting property that it converges in distribution to the time varying Bayes posterior described above. This jump-diffusion Markov process $\{X(s), s \geq 0\}$ samples the posterior distribution $\pi_t(\vec{x})$

defined over the full parameter space $\mathcal{X}_t$, i.e. the time samples of the process visit the elements in the parameter space $\mathcal{X}_t$ according to the posterior distribution. More importantly, the samples generated by this Markov process can be utilized to empirically reconstruct the posterior distribution or to evaluate any mean associated with it.

The estimation process is, in effect, a search for the features representing an observed scene. These features can be of continuous type, e.g. the target positions in $\Re^3$ and orientations in $\mathcal{M}(3)$ or of discrete types, e.g. the number of targets $M \in \aleph$ and target-type $a \in \mathcal{A}$. The two constituents of this search, i.e. *Jump* and *Diffusion*, contribute to the search for discrete and continuous features, respectively.

1.4 Results and Contribution

Here, we present a new random sampling algorithm for estimating the characteristics of moving signal sources. The method of estimation is to derive a single posterior distribution over the space $\mathcal{X}_t$ and then sample it via a Markov process $X(s)$ which satisfies jump-diffusion dynamics. This approach solves the full Bayesian problem as, in theory, no approximation is necessary. It is based on the work of Grenander and Miller [11, 13, 12] who have described a new class of sampling algorithms for a wide variety of applications including image analysis, crystallography and stochastic language models. These algorithms involve stochastic search over well defined parameter spaces following the Bayesian measures on these spaces.

An implementation of this algorithm for estimating a single track scene is presented. The algorithm was jointly implemented using a Silicon Graphics workstation for data generation and visualization, and a massively parallel 4096 processor SIMD

DECmpp machine for implementing the tracking-recognition algorithm. It includes generating a parameterized target path using the Silicon Graphics' flight simulator which forms the true configuration. Using this path the simulated data are generated for both the tracking and imaging sensors. The generated data are then used in the estimation process to obtain the *MMSE* estimates of the actual flight.

Chapter 2 lists the set of parameters completely describing an observed scene and outlines the estimation problem. The posterior distribution on the parameter space is derived in chapter 3 by describing the dynamics based prior and observed data likelihoods. An estimation algorithm based on jump-diffusion processes is presented in chapter 4 along with the important theoretical results supporting the algorithm. Chapter 5 describes the use of a jump-diffusion algorithm for estimation of a single track configuration. It involves the implementation across machines such as the Silicon Graphics workstation and the DECmpp 12000 SIMD machine connected through data transfer on a high speed network. To emphasize the use of this algorithm for multi-target scenes a two-dimensional (2-D) scenario is explained in chapter 6. This involves tracking the direction of arrival (DOA) of multiple targets using uniform linear array of isotropic sensors with motion restricted to the 2-D plane containing the array. The problem setup, parameterization and implementation are presented along with the results for multi-target scene estimation. Some ideas for future work are explored in chapter 7.

2. Scene Parameterization

2.1 Parameter Set

We use the global shape models and pattern theoretic approach introduced by Grenander [20, 21] to analyze complex scenes. As the basic building blocks of the hypotheses we define a subset of generators $\mathcal{G}^o$, which contains each target type $a \in \mathcal{A}$ placed at the origin of the inertial reference frame at a fixed orientation and unit scale. The fundamental variability in target spaces is accommodated by applying the transformations $T(\vec{\phi}), T(\vec{p}), T(s)$ to the templates $g^o \in \mathcal{G}^o$ according to

$$T(\vec{\phi}) : \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi_1 & sin\phi_1 \\ 0 & -sin\phi_1 & cos\phi_1 \end{bmatrix} \begin{bmatrix} cos\phi_2 & 0 & sin\phi_2 \\ 0 & 1 & 0 \\ -sin\phi_2 & 0 & cos\phi_2 \end{bmatrix} \times \begin{bmatrix} cos\phi_3 & sin\phi_3 & 0 \\ -sin\phi_3 & cos\phi_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{2.1}$$

$$T(\vec{p}) : \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \rightarrow \begin{bmatrix} x_1 + p_1 \\ x_2 + p_2 \\ x_3 + p_3 \end{bmatrix}, \tag{2.2}$$

$$T(s) : \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \cdot \rightarrow \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & s \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \tag{2.3}$$

where $\vec{\phi}$ is the triple of rotation angles (pitch, roll and yaw), $\vec{p}$ is the translation vector, and s is the scale parameter. These parameterized transformations operate on the templates from $\mathcal{G}^o$ generating the full set of elements $\mathcal{G}$. The observed scene at any time is modeled as a set of generators $G = \{g(1), g(2), .., g(M)\}$, each generator $g(m) \in \mathcal{G}$. Then, $\{a, \vec{\phi}, \vec{p}, s\}$ parameterize the representation of all possible targets generated from these transformations. Figure 2.1 shows one of the 3-D ideal targets used for all the simulations presented here. The left panel shows a rendering of the target generator $g^o \in \mathcal{G}^o$ at the origin, the right panel showing the result of applying one of the rotation transformations resulting in $g \in \mathcal{G}$.

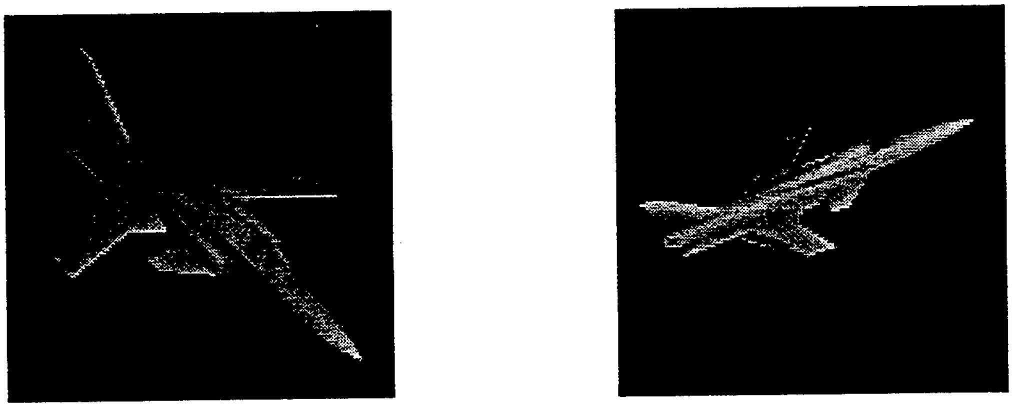


Figure 2.1: 3-D target generator $g \in \mathcal{G}^o$ at the origin (left panel) and after applying a rotation transformation (right panel).

2.2 Parametric Space

The set parameterizing the Bayes posterior becomes the set of parameters specifying the similarity transformations, as well as the airplane type. Define the space containing orientations $\vec{\phi}$ as the three dimensional torus $\mathcal{M}(3) \equiv [0, 2\pi]^3$ with $0, 2\pi$ identified. The position vector $\vec{p}$ belongs to $\Re^3$ with the scale parameter belonging to $\Re_+$. Then associated with each target or generator $g \in \mathcal{G}$ at any time τ is a parameter set $x(\tau) = \{a(\tau), \vec{p}(\tau), \vec{\phi}(\tau), s(\tau)\} \in \mathcal{M}(3) \times \Re^3 \times \mathcal{A} \times \Re_+$, where $|\mathcal{A}| = |\mathcal{G}^\circ|$ the number of different target types.

A pattern will be constructed for the representation of the multiple track scenes with varying track lengths. We are interested in tracking-recognition in non-cooperative environments in which the m^{th} object appears and disappears at random times $t_1^{(m)}, t_1^{(m)} + t^{(m)}$ with its stay given by the interval $\mathcal{T}^{(m)} = [t_1^{(m)}, t^{(m)} + t_1^{(m)}]$. Clearly, $t_0 \leq t_1^{(m)} \leq t^{(m)} + t_1^{(m)} \leq t$, for the observation interval $[t_0, t]$. Define $x^{(m)}(\tau)$ to be the set of parameters associated with the target m at time τ given by $\{\vec{p}^{(m)}(\tau)\, , \vec{\phi}^{(m)}(\tau)\, , a^{(m)}(\tau)\, , s^{(m)}(\tau)\}$. For the observation period $[t_0, t]$ the parameter vector associated with the complete m^{th} track given $\mathcal{T}^{(m)}$ is

$$\{x^{(m)}(\tau) : \tau \in \mathcal{T}^{(m)}\} \in \left(\mathcal{M}(3) \times \Re^3 \times \Re_+ \times \mathcal{A}\right)^{\mathcal{T}^{(m)}}$$

In real situations with discrete observations, the tracks get discretized to observation times $1, 2, \ldots . t \in \aleph$ (assume $t_0 = 1$ for simplicity). We denote the discrete parameters with the same symbols except now they belong to discrete sets, i.e. $t_1^{(m)}, t^{(m)}, t \in \aleph$ and $\mathcal{T}^{(m)} = \{t_1^{(m)} + 1, \ldots, t_1^{(m)} + t^{(m)}\}$. Hence the parameter set associated with the discretized track $\{x^{(m)}(k); k \in \mathcal{T}^{(m)}\}$ given $\mathcal{T}^{(m)}$ is an element of $(\mathcal{M}(3) \times \Re^3 \times \Re_+ \times \mathcal{A})^{t^{(m)}}$. Since the dwell time of the target, given by $\mathcal{T}^{(m)}$, is unknown a-priori its associated parameter set is an element of

$$\bigcup_{t^{(m)}=1}^{t} \left(\mathcal{M}(3) \times \Re^3 \times \Re_+ \times \mathcal{A}\right)^{t^{(m)}} \times \aleph \, .$$

The parameter vector for an M-track scene becomes the collection of each of the single track parameter sets, element of $\mathcal{X}_t(M)$ according to

$$\begin{aligned} \vec{x}_t(M) &= \bigcup_{m=1}^{M} \{x^{(m)}(k) : k \in \mathcal{T}^{(m)}\} \\ &\in \mathcal{X}_t(M) \equiv \prod_{m=1}^{M} \left(\bigcup_{t^{(m)}=1}^{t} \left(\mathcal{M}(3) \times \Re^3 \times \Re_+ \times \mathcal{A}\right)^{t^{(m)}} \times \aleph \right) . \end{aligned}$$

For later convenience we also define $\vec{p}_t(M)$ to be the vector having elements the position components for all tracks and $\vec{\phi}_t(M)$ to be the vector containing orientation components of all tracks. Since M is unknown we define the complete configuration space $\mathcal{X}_t$ over which the estimation is performed as

$$\mathcal{X}_t = \bigcup_{M=0}^{\infty} \mathcal{X}_t(M) \, .$$

The estimation problem is to estimate the individual configurations as well as the number M. In the work presented, only rigid transformations are used with $s = 1$.

3. Bayesian Posterior

We take a Bayesian approach for solving the estimation problem by defining a posterior probability on the parameter space $\mathcal{X}_t$. As the posterior distribution is proportional to the product of the prior distribution and the observed data likelihood we first derive a prior measure on the parameter space followed by a model for the data generation which determines the likelihood term.

The prior measure encodes our *a-priori* information about the parameters to be estimated. In particular this knowledge can come from, say, the airplane dynamics, or some previous knowledge of target type and number of targets. In the work presented here, we utilize the set of Newton's second law based equations governing airplane motion to generate a prior distribution on the airplane paths. The prior on the orientation parameters is based on the von-Mises density. There are two types of data sets used here: the tracking data collected by a cross-array of isotropic sensors and the imaging data generated by optical sensing radar. The likelihood of the tracking data follows from the standard narrowband signal model, first proposed by Schmidt[14], whereas the imaging data are simply given by the far field projection of the actual 3-D object.

It needs to be emphasized that in real time estimation problems like these, where the data set is augmented at every observation time, the posterior distribution changes with time t, and is an explicit function of t denoted by $\pi_t(\cdot)$, $t \in \aleph$.

Therefore, in this Bayesian approach the estimates are generated at any given time conditioned on the data accumulated up to that time.

3.1 Prior Density on Parameter Space $\mathcal{X}_t$

The formulation of the prior measure on the airplane positions $\vec{p}(s)$ is based on the equations of motion governing the airplane's flight.

3.1.1 Analysis of Airplane Motion

First, we derive the prior for a single target case by considering its underlying continuous motion. This derivation follows the description in [17], where the equations describing target dynamics are utilized to form a prior measure on the airplane positions $\vec{p}(s)$. These dynamics are easily expressed using the target velocities projected along the body-fixed axes, called the body-frame velocities $\vec{v}(s)$, as shown in Figure 3.1. Since the tracking array responds to the inertial positions of the target, we use the standard transformation to relate body-frame velocities with inertial frame

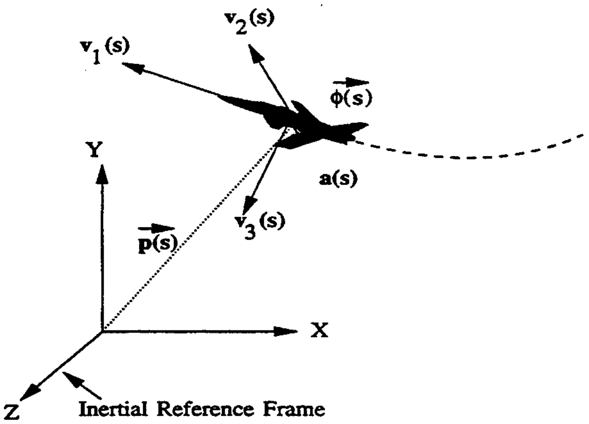


Figure 3.1: The observed target located at position $\vec{p}(s)$, oriented at $\vec{\phi}(s)$ with body frame velocities $\vec{v}(s)$.

positions given by

$$\vec{p}(s) = \int_{t_0}^{s} \Phi(\tau)\vec{v}(\tau)d\tau + \vec{p}(t_0), \tag{3.1}$$

where $\vec{p}(t_0)$ is assumed known and $\Phi(\tau)$ is a function of the Euler angles $\vec{\phi}(\tau)$ given by

$$\Phi(\tau) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi_1(\tau) & sin\phi_1(\tau) \\ 0 & -sin\phi_1(\tau) & cos\phi_1(\tau) \end{bmatrix} \begin{bmatrix} cos\phi_2(\tau) & 0 & sin\phi_2(\tau) \\ 0 & 1 & 0 \\ -sin\phi_2(\tau) & 0 & cos\phi_2(\tau) \end{bmatrix}$$

$$\times \begin{bmatrix} cos\phi_3(\tau) & sin\phi_3(\tau) & 0 \\ -sin\phi_3(\tau) & cos\phi_3(\tau) & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

which is same as the rotation matrix in Eqn 2.1. In general, $\Phi(\tau)$ converts any vector in the body frame of reference to the corresponding vector in the inertial frame. The rotation dynamics are described in terms of the angular velocities $\vec{q}(s)$ which are the known functions of the Euler angles $\phi(s)$ and their rates of change $\dot{\vec{\phi}}(s)$, given by Eqn 9.4 ([22]).

Following the rigid body analysis in [17], we neglect the earth's curvature, motion and wind effects. Then, the linear velocities $\vec{v}(s)$ and the angular velocities $\vec{q}(s)$ satisfy the following set of differential equations,

$$\begin{aligned} \dot{v}_1(s) - q_3(s)v_2(s) + q_2(s)v_3(s) &= f_1(s)\ , \\ \dot{v}_2(s) + q_3(s)v_1(s) - q_1(s)v_3(s) &= f_2(s)\ , \\ \dot{v}_3(s) - q_2(s)v_1(s) + q_1(s)v_2(s) &= f_3(s)\ , \\ I_1\dot{q}_1(s) - (I_2 - I_3)q_2(s)q_3(s) &= \Gamma_1(s)\ , \\ I_2\dot{q}_2(s) - (I_3 - I_1)q_1(s)q_3(s) &= \Gamma_2(s)\ , \end{aligned}$$

$$I_3\dot{q}_3(s) - (I_1 - I_2)q_2(s)q_1(s) \;=\; \Gamma_3(s) \,,$$

where $\vec{f}(s) = [f_1(s)\ f_2(s)\ f_3(s)]$ is the vector of applied translational forces, $\vec{I} = [I_1\ I_2\ I_3]$ is the vector of rotational inertias, and $\vec{\Gamma}(s) = [\Gamma_1(s)\ \Gamma_2(s)\ \Gamma_3(s)]$ is the vector of applied torques. The first three equations describe the airplane's translational motion while the next three describe its rotational motion. We propose to use these equations to derive prior densities on the translational and rotational motions. In the work presented here, we use only the first three equations for deriving priors on the positions using simple Markov priors for the rotational parameters instead of the last three equations. In vector form the first three equations become

$$\dot{\vec{v}}(s) + A_1(\vec{\phi}(s), \dot{\vec{\phi}}(s))\vec{v}(s) = \vec{f}(s) \tag{3.2}$$

where

$$A_1(\vec{\phi}(s), \dot{\vec{\phi}}(s)) = \begin{bmatrix} 0 & -q_3(s) & q_2(s) \\ q_3(s) & 0 & -q_1(s) \\ -q_2(s) & q_1(s) & 0 \end{bmatrix} ,$$

the angular velocities $\vec{q}(s)$ being determined by the Euler angles $\vec{\phi}(s)$ and their rates of change. Since the eigen values of $A_1(\vec{\phi}(s), \dot{\vec{\phi}}(s))$ lie on the imaginary axis, this system is not stable. Following standard treatments (see [22] for example), we add a term $G\vec{v}(s)$ to the force vector, which provides a linear stabilizing feedback to the system. A simple diagonal gain matrix G is used, with the resulting system equation given by

$$\dot{\vec{v}}(s) + A(\vec{\phi}(s), \dot{\vec{\phi}}(s))\vec{v}(s) = \vec{f}(s), \tag{3.3}$$

where $A(\vec{\phi}(s), \dot{\vec{\phi}}(s)) = A_1(\vec{\phi}(s), \dot{\vec{\phi}}(s)) - G$. This linear vector differential equation is characterized by the time-varying parameter matrix $A(\vec{\phi}(s), \dot{\vec{\phi}}(s))$ which depends on the rotational motion of the airplane. The prior is induced following the approach in Amit et al. [23] and used in [17] by assuming the forcing function to be a white process with fixed spectral density σ_0^2. As the equations are linear in velocity vector this induces a Gaussian prior on $\vec{v}(s)$ conditioned on the Euler angles with the covariance determined by solving the differential equation 3.3 as shown in Appendix 2. This covariance function is given by (Eqn 9.5),

$$\mathcal{K}_v(s_1, s_2) = \sigma_0^2 \int_{t_0}^{min(s_1,s_2)} [e^{\int_{t_1}^{s_1} A(\vec{\phi}(\tau), \dot{\vec{\phi}}(\tau))d\tau}][e^{-\int_{t_1}^{s_2} A(\vec{\phi}(\tau), \dot{\vec{\phi}}(\tau))d\tau}]^\dagger dt_1 \ . \qquad (3.4)$$

The covariance of the Gaussian density on inertial positions becomes

$$\mathcal{K}_p(s_1, s_2) = \int_{t_0}^{s_1} \int_{t_0}^{s_2} \Phi(\tau_1) \mathcal{K}_v(\tau_1, \tau_2) \Phi^\dagger(\tau_2) d\tau_1 d\tau_2 \ . \qquad (3.5)$$

Notice that this covariance function is parameterized by the sequence of airplane orientations thereby demonstrating the fundamental connections between the tracking and recognition algorithms. For M targets, the prior density is the product of individual Gaussian densities of each target. The Gibbs' potential associated with that prior density on the set of inertial positions can be written as

$$P_t(\vec{p}_t(M)) = -\frac{1}{2\sigma_1^2} \vec{p}_t(M)^\dagger K_p^{-1} \vec{p}_t(M), \qquad (3.6)$$

where K_p (obtained using covariance function $\mathcal{K}_p$) is the $3n_M \times 3n_M$ covariance matrix of the position vector $\vec{p}_t(M)$, and $n_M = \sum_{m=1}^{M} t^{(m)}$, the total number of track-segments in $\vec{x}_t(M)$.

3.1.2 Prior on Rotational Motion

In this work, we utilize a von-Mises prior on the orientation angles $\vec{\phi} = [\phi_1, \phi_2, \phi_3] \in \mathcal{M}(3)$ (pitch, roll and yaw). For simplification, it is assumed that the three angles are statistically independent of each other (even though they are known to be related). For circular parameters the von-Mises density is analogous to the normal distribution on the real line [24, 25]. The Gibbs' potential associated with the prior on the rotational motion of M targets is given by

$$P_t(\vec{\phi}_t(M)) = \kappa \left(\sum_{m=1}^{M} \sum_{(k \in \mathcal{T}^{(m)})} \sum_{i=1}^{3} cos(\phi_i^{(m)}(k) - \phi_i^{(m)}(k-1)) \right) \tag{3.7}$$

where $\kappa > 0$ is called the concentration parameter. The prior potential on the complete parameter space $\mathcal{X}_t$ given M becomes

$$P_t(\vec{x}_t(M)) = P_t(\vec{p}_t(M)) + P_t(\vec{\phi}_t(M))$$

3.2 Data Likelihood

In this section, we derive the likelihood of collected data conditioned on a given set of parameters. There are two sensor types in our problem, a *tracking sensor*, consisting of an array of passive sensors and a range radar, and a *high-resolution imaging* sensor.

3.2.1 Tracking

For azimuth-elevation coordinate tracking, a cross array of isotropic sensors (Figure 3.2) is assumed as in [15, 16, 17, 18] using the standard narrowband signal model

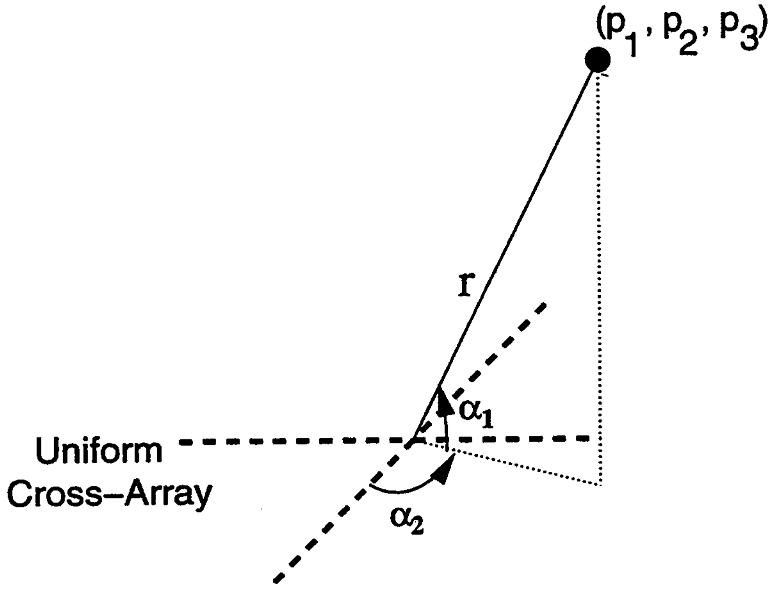


Figure 3.2: The figure displays the cross array of isotropic sensors at half wavelength spacing, which observes the angular location of the target.

developed in [14]. Accordingly the signal arriving at the sensors is assumed to be in a relatively small band in the frequency spectrum, such that the signal amplitudes remain approximately constant as wavefronts traverse the array with only difference among the signals reaching different elements being due to the relative phase lags. Depending on the geometry of the sensor arrangement, determining the array-manifold, these phase lags are known functions of the source locations (Appendix 1). The amplitudes of the arriving signals $\vec{s}(k)$ can be modeled either as unknown deterministic values or as random variables. In this work, we assume the deterministic signal model in which the measurements $\vec{y}_1(k)$ are Gaussian distributed with mean given by the signal component. Accordingly, the expression for sensor response to M signal sources at locations $\vec{p}^{(1)}(k), .., \vec{p}^{(M)}(k)$ with amplitudes

$\vec{s}(k) = [s^1(k) \ ,.., s^M(k)]$ is

$$\vec{y}_1(k) = \sum_{m=1}^{M} d(\vec{p}^{(m)}(k)) 1_{\mathcal{T}^{(m)}}(k) s^{(m)}(k) + \vec{n}_1(k) \ , \tag{3.8}$$

where $\vec{n}_1(k)$ is a 0-mean complex Gaussian noise vector of the Goodman class with covariance $\sigma_1^2 I$, $1_{\mathcal{T}^{(m)}}(k)$ selects the targets that contribute in the signal at time k, and $\vec{d}(\vec{p}^{(m)}(k))$ is the vandermonde direction vector corresponding to m^{th} target, described in Appendix 1. In a general problem, the signal amplitudes are also to be estimated from the collected data. But we focus on the estimation of the target positions by assuming the signal amplitudes to be known. The ambient noise surrounding the sensor elements is modeled as a white Gaussian process, i.e. the noise samples added to the signal at different sensors or different times are uncorrelated.

The set of tracking data collected up to time t is given by $I_t^{\mathcal{P}}(1) \equiv \{\vec{y}_1(k) : k \in \{1,..,t\}\}$, and the likelihood of these data has the Gibbs' potential

$$L_t^1(\vec{x}_t(M)) \ = \ -\frac{1}{\sigma_1^2} \sum_{k=1}^{t} |\vec{y}_1(k) - \sum_{m=1}^{M} d(\vec{p}^{(m)}(k)) 1_{\mathcal{T}^{(m)}}(k) s^{(m)}(k)|^2 \ .$$

3.2.2 Imaging

While the statistical models for high-resolution radar imaging are being incorporated in this problem by others ([26, 27, 28, 29, 30, 31]), all of the results shown here are based on an optical imaging system. In this system, the data are a sequence of 2-D images resulting from projecting the target onto the focal plane of the imaging sensor; i.e., the deformation of the imaging process of ideal targets is assumed here to be far field orthographic projection. This projection $\mathcal{P}(\cdot)$ defines the deterministic operation of imaging the scene containing multiple objects,

$$\mathcal{P} : \Re^{\mathcal{V}} \rightarrow \Re^{\mathcal{L}} ,$$

where $\mathcal{V}$ is the imaged space and $\mathcal{L}$ is the discrete lattice on which the 3D volume is projected. This projection is described as follows: we define a scene, or a configuration of multiple generators $g(1), g(2), ..g(M)$ to be the generators placed and oriented according to their associated parameter vector. Then the volume $\mathcal{V}$ containing the generators is projected onto $\mathcal{L}$ using far field orthographic assumptions as shown in the Figure 3.3. Since the parameter set $\vec{x}_t(M)$ completely determines the imaged volume, we can also write the projection as an operation from the parameter space to $\Re^{\mathcal{L}}$, i.e. $\mathcal{P} : \mathcal{X}_t \rightarrow \Re^{\mathcal{L}}$.

We choose the lattice to be an $\mathcal{L} = 64 \times 64$ array implying the imaging data at time k form the set of 64×64 grey scale pixel values $\vec{y}_2(k) \in [0, 255]^{64\times 64}$. For simulations, $\mathcal{P}$ is implemented using the Silicon Graphics imaging system. For the implementation presented here, a Gaussian noise model was used, with the measured data for the set of M targets having mean given by the projection of M targets. For the scene containing M targets the likelihood potential becomes

$$L_t^2(\vec{x}_t(M)) = -\frac{1}{2\sigma_2^2} \sum_{k=1}^{t} ||\vec{y}_2(k) - \mathcal{P}(\vec{x}_t(M))||^2 \tag{3.9}$$

where $|| \cdot ||$ represents matrix 2-norm, and σ_2^2 is the noise power.

The imaging data set up to time t are given by $I_t^{\mathcal{P}}(2) \equiv \{\vec{y}_2(k) : k \in \{1, ., t\}\}$, and the complete data set becomes $I_t^{\mathcal{P}} = \{I_t^{\mathcal{P}}(1), I_t^{\mathcal{P}}(2)\}$. The combined data likelihood

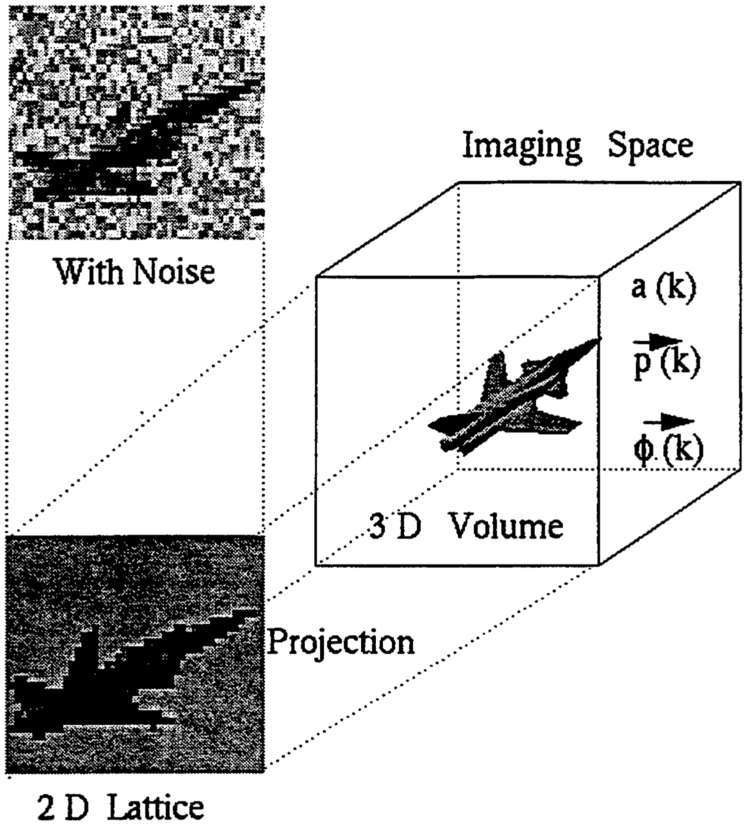


Figure 3.3: The projection transformation $\mathcal{P}$ converting a 3D volume into a 2D image on a discrete lattice $\mathcal{L}$ for a single target.

has potential

$$
\begin{aligned}
L_t(\vec{x}_t(M)) &= L_t^1(\vec{x}_t(M)) + L_t^2(\vec{x}_t(M)) \\
&= -\frac{1}{\sigma_1^2}\sum_{k=1}^{t} |\vec{y}_1(k) - \sum_{m=1}^{M} d(\vec{p}^{(m)}(k))1_{\mathcal{T}^{(m)}}(k)s^{(m)}(k)|^2 \\
&- \frac{1}{2\sigma_2^2}\sum_{k=1}^{t}\{||\vec{y}_2(k) - \mathcal{P}(\vec{x}_t(M))||^2\}
\end{aligned}
$$

Shown in Figure 3.4 are four samples of the imaging data for a single target scene. It shows the ideal projected onto a 2-D lattice with additive noise at four

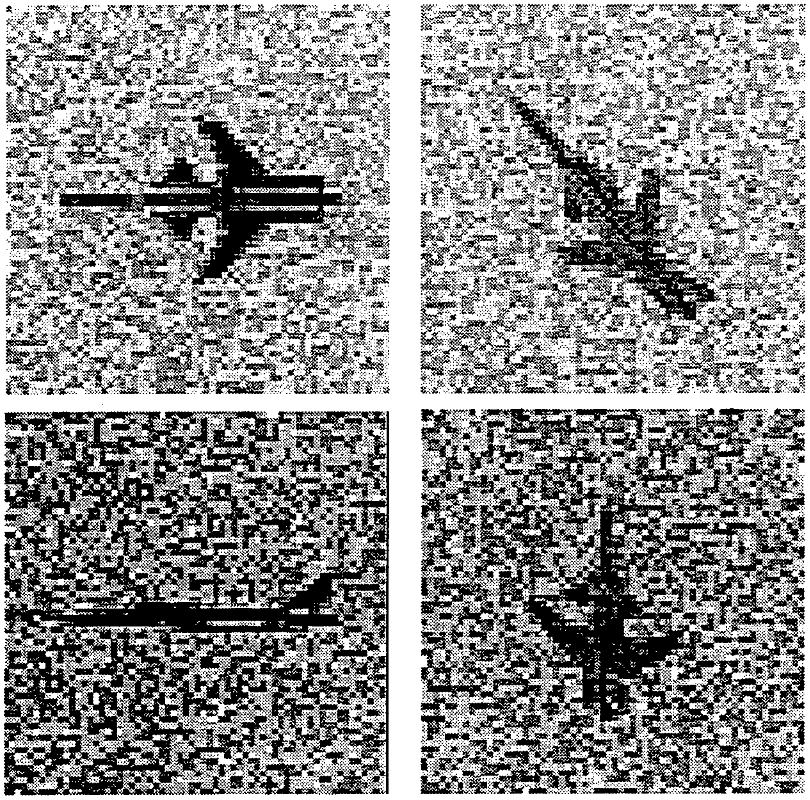


Figure 3.4: The figure shows the target projected onto a 64 × 64 2-D lattice with additive noise at four different time instants.

different time instants. Figure 3.5 shows the spatial power spectrum of the tracking data generated using the *minimum variance distortionless response* (MVDR) [35] spectral analysis at four instants of time, plotted in the azimuth-elevation plane (bright is low power, dark is high power).

Remark: For observing the range locations of the targets, a range radar is assumed with the observations modeled as normally distributed with mean $|\vec{p}(k)|$, the 2-norm of the position vector at time k.

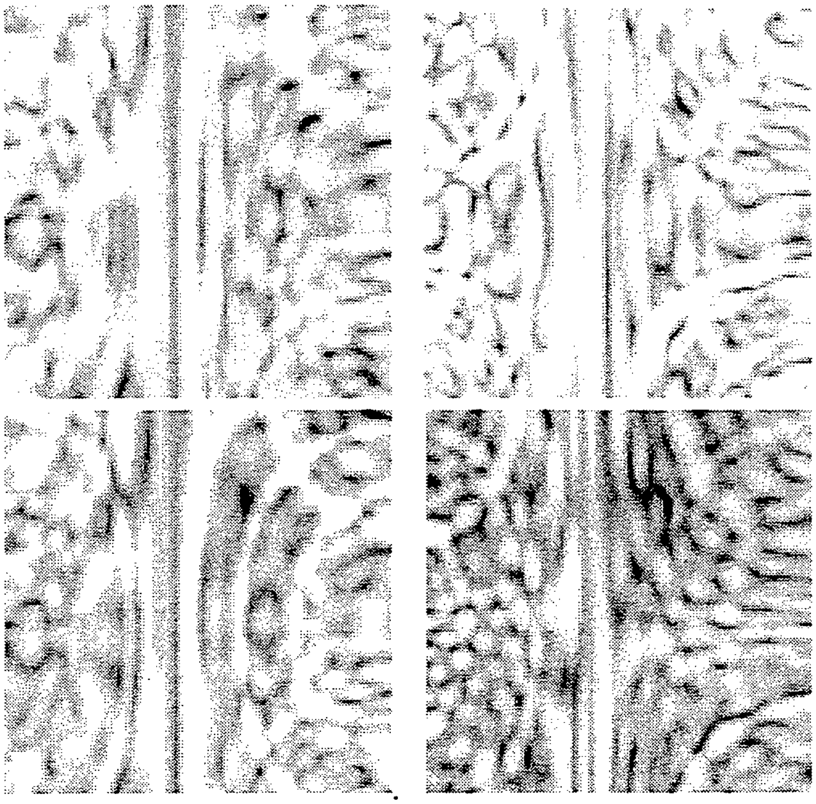


Figure 3.5: The figure shows the azimuth-elevation spatial power spectrum of the narrowband tracking data generated via MVDR method at four different instants of time (bright is low power, dark is high power).

3.3 Bayesian Posterior

The posterior distribution is obtained as the product of the data likelihood and the prior distribution using Bayes' rule. The posterior distribution in Gibbs' form becomes

$$\begin{aligned}\pi_t(\vec{x}_t(M)|M) &= \frac{1}{\mathcal{Z}(M)}e^{-E_t(\vec{x}_t(M)|M)} \\ &= \frac{1}{\mathcal{Z}(M)}e^{-(P_t(\vec{x}_t(M))+L_t(\vec{x}_t(M)))}\end{aligned},$$

where $L_t(\vec{x}_t(M))$ is the potential associated with data likelihood, and $P_t(\vec{x}_t(M))$ is the potential associated with the prior distribution on the parameter space $\mathcal{X}_t(M)$.

So far we have defined a family of posterior distributions $\mu_t(\cdot|M)$ each associated with a subspace $\mathcal{X}_t(M)$ such that

$$\mu_t(d\vec{x}_t(M)|M) = \frac{1}{\mathcal{Z}(M)} e^{-E_t(\vec{x}_t(M)|M)} d\vec{x}_M$$

where $d\vec{x}_M$ is the appropriate Lebesgue measure associated with the space in which $\vec{x}_t(M)$ is an element. The distribution over the complete parameter space $\mathcal{X}_t$ is defined as the convex combination, i.e. $\mu_t(\cdot) \equiv \sum_{M=0}^{\infty} p(M)\mu_t(\cdot|M)$, where $\sum_{M=0}^{\infty} p(M) = 1$ is defined in the sense that for any $\mathcal{S} \subset \mathcal{X}_t$,

$$\mu_t(\mathcal{S}) = \sum_{M=0}^{\infty} p(M)\mu_t\left(\mathcal{S} \bigcap \mathcal{X}_t(M)|M\right) .$$

Having derived a posterior measure over the complete parameter space we now describe an estimation procedure based on the jump-diffusion random sampling algorithm.

4. Estimation Through Random Sampling

4.1 Random Sampling

Random sampling from a probability measure over a state space refers to drawing the elements from that space according to that fixed probability measure. The samples are generated via a Markov process which visits the elements of the state space with the frequencies proportional to that probability measure. This implies that the empirical averages generated from the samples of the Markov process converge to their conditional means under the given density.

4.1.1 Why Random Sampling

The estimates are obtained on the basis of *MMSE* criterion which minimizes the cost function $\mathcal{E}\{(\vec{x}_t(M) - \hat{\vec{x}}_t(M))^2 | I_t^{\mathcal{D}}\}$. So the problem is to search for $\hat{\vec{x}}_t(M)_{MS}$ such that

$$\hat{\vec{x}}_t(M)_{MS} = arg \min_{\hat{\vec{x}}_t(M) \in \mathcal{X}_t} \mathcal{E}\{(\vec{x}_t(M) - \hat{\vec{x}}_t(M))^2 | I_t^{\mathcal{D}}\} ,$$

where $\mathcal{E}$ stands for the expectation operator. It is well known that this minimizer is given by the conditional mean ([32]) under the posterior density, i.e.

$$\hat{\vec{x}}_t(M)_{MS} = \mathcal{E}\{\vec{x}_t(M) | I_t^{\mathcal{D}}\}$$

In most problems of practical relevance it is difficult to analytically solve for this conditional mean because of the complicated posterior densities involved. Therefore we define a random sampling mechanism which draws samples from the posterior density such that their averages converge to the conditional mean. This sampling mechanism is based on the jump-diffusion processes.

4.2 Jump-Diffusion Sampling Algorithm

The sampling process is essentially a search for the features which best conform to the given data set. These features can be of discrete nature, e.g. the number of targets, and target type or they can live in continuous spaces, e.g. the target positions and orientations. Accordingly, there are two components in the discovery process which account for these two kinds of feature variabilities. The jump process involves discrete moves over non-connected subspaces searching for the discrete features while the diffusion component performs continuous stochastic gradients estimating the continuous features.

Our approach is to construct a jump-diffusion Markov process, following the analysis outlined in [11], having the limiting property that it converges in distribution to the Bayes posterior. This implies that the time samples of the Markov process visit the configurations with high probability more often. Following the *jump-diffusion* dynamics the process (i) on random exponential times jumps from one of the countably infinite set of subspaces to another estimating discrete parameters, and (ii) between jumps it performs diffusion following the S.D.E.'s appropriate for subspace it is in.

As mentioned previously, the posterior density changes at each observation time due to the addition of one more data sample to the data set. Therefore, at any given time t, the sampling process generates samples from the posterior density having the Gibbs energy $E_t(\vec{x}_t(M))$. The jump-diffusion Markov process $\{X(s), s \geq 0\}$ samples from the posterior density $\pi_t(\vec{x}_t(M))$ defined over the full parameter space $\mathcal{X}_t$ as follows.

To simplify the following analysis we introduce some additional notation. Define a deletion operator $\wp$ for deleting elements from the present configuration such that $\wp_T^{(j)}$ deletes the j^{th} track while $\wp_S^{(j)}$ removes the last track segment of the j^{th} track, i.e.,

$$\begin{aligned}
\wp_T^{(j)} \ &: \ \Big(\mathcal{M}(3) \times \Re^3 \times \mathcal{A}\Big)^{(\sum_{m=1}^{M} t^{(m)})} \times \aleph^M \rightarrow \\
&\qquad\qquad\qquad \Big(\mathcal{M}(3) \times \Re^3 \times \mathcal{A}\Big)^{(\sum_{m\neq j}^{M} t^{(m)})} \times \aleph^{M-1} \ , \\
\wp_S^{(j)} \ &: \ \Big(\mathcal{M}(3) \times \Re^3 \times \mathcal{A}\Big)^{(\sum_{m=1}^{M} t^{(m)})} \times \aleph^M \rightarrow \\
&\qquad\qquad\qquad \Big(\mathcal{M}(3) \times \Re^3 \times \mathcal{A}\Big)^{(\sum_{m=1}^{M} t^{(m)})-1} \times \aleph^{M} \ .
\end{aligned}$$

Also, $\oplus_j$ stands for the addition of elements in the present configuration at j^{th} track, i.e. $\vec{x}_t(M) \oplus_j \vec{y}_t(1)$ represents an $M+1$ track configuration formed by adding $\vec{y}_t(1)$ to $\vec{x}_t(M)$ at the j^{th} location. Similarly $\vec{x}_t(M) \oplus_j y^{(j)}$ signifies addition of a segment to the j^{th} track of $\vec{x}_t(M)$.

4.2.1 Jump Process

The jump process deals with the assignment of tracks and the choice of model order in two ways. First, the individual tracks are developed by probabilistically placing the track-segments sequentially in the associated track configuration. Secondly, the jump process moves among the subspaces of variable numbers of tracks via the addition and deletion of tracks.

For hypothesizing the existence of new tracks and the disappearance of faulty track hypotheses as well as growing and shrinking tracks, we use classical ideas associated with birth-death processes, where a birth corresponds to newly hypothesized track/track-segment in the scene and a death the removal of one. These births/deaths are two of a family of *simple jump moves*, others corresponding to splitting and fusion of tracks, with the simple moves transforming one model to another. The jump transformations are applied discontinuously and drawn probabilistically from a rich family of transformations. The jumps take one model into another satisfying the condition that given any two models of the dimensions M, M', it should be possible to find a finite chain of transitions leading from one to the other.

We allow only those jump moves which result in the following types of transformations through parameter space

$$\begin{aligned}
\text{Addition of track} &: \vec{x}_t(M) \;\rightarrow\; \vec{x}_t(M) \oplus_j \vec{y}_t(1), \\
\text{Addition of track-segment} &: \vec{x}_t(M) \;\rightarrow\; \vec{x}_t(M) \oplus_j y^{(j)}\ , \\
\text{Deletion of track} &: \vec{x}_t(M) \;\rightarrow\; \wp_T^{(j)} \vec{x}_t(M)\ , \\
\text{Deletion of track-segment} &: \vec{x}_t(M) \;\rightarrow\; \wp_S^{(j)} \vec{x}_t(M)\ .
\end{aligned}$$

It should be noted that the addition of only unit length tracks is allowed. Let $\mathcal{T}^1(\vec{x}_t(M))$ be the set of configuration types that can be reached from $\vec{x}_t(M)$ in one jump move, i.e.

$$\mathcal{T}^1(\vec{x}_t(M)) = \{\vec{x}_t(M) \oplus_j \vec{y}_t(1), \vec{x}_t(M) \oplus_j y^{(j)}, \wp_T^{(j)}\vec{x}_t(M), \wp_S^{(j)}\vec{x}_t(M)\},$$

$\mathcal{X}_t(\mathcal{T}^1(\vec{x}_t(M)))$ being the space containing the configurations of these types. The discrete jump moves are performed on the basis of following jump parameters (Appendix 3) defined for $a, b \in \mathcal{X}_t$ as:

- $q(a, db)$: the transition measure from the configuration a to an infinitesimal neighborhood of b.
- $q(a)$: the intensity of jumping out of configuration a, $q(a) = \int_{\mathcal{X}_t(\mathcal{T}^1(a))} q(a, db)$.
- $Q(a, db)$: the transition probability from a to db. $Q(a, db) = \frac{q(a,db)}{\int_{\mathcal{X}_t(\mathcal{T}^1(a)} q(a,db))}$.

The transition measures for the feasible jump moves are given by

$$\begin{aligned}
q(\vec{x}_t(M), d\vec{y}_t(M+1)) &= \sum_{j=1}^{M+1} q_T^b(\vec{x}_t(M), \vec{y}(M+1))\delta_{\vec{x}_t(M)}\Big(d(\wp_T^{(j)}\vec{y}(M+1))\Big)d\vec{y}(1)\ , \\
q(\vec{x}_t(M), d\vec{y}_t(M)) &= \sum_{j=1}^{M} q_S^b(\vec{x}_t(M), \vec{y}_t(M))\delta_{\vec{x}_t(M)}\Big(d\Big(\wp_S^{(j)}\vec{y}_t(M)\Big)\Big)dy \\
&+ \sum_{j=1}^{M} q_S^d(\vec{x}_t(M), \vec{y}_t(M))\delta_{\wp_S^{(j)}\vec{x}_t(M)}(d\vec{y}_t(M))\ , \\
q(\vec{x}_t(M), d\vec{y}_t(M-1)) &= \sum_{j=1}^{M} q_T^d(\vec{x}_t(M), \vec{y}_t(M-1))\delta_{\wp_T^{(j)}\vec{x}_t(M)}(d\vec{y}_t(M-1)) \qquad (4.1)
\end{aligned}$$

and the intensity of jumping out of $\vec{x}_t(M)$ is given by

$$q(\vec{x}_t(M)) = \sum_{j=1}^{M+1} \int_{\mathcal{X}_t(1)} q_T^b\Big(\vec{x}_t(M), \vec{x}_t(M) \oplus_j \vec{y}_t(1)\Big)d(\vec{y}_t(1))$$

$$
\begin{aligned}
&+\sum_{j=1}^{M} q_T^d\Big(\vec{x}_t(M), \wp_T^{(j)}\vec{x}_t(M)\Big) \\
&+\sum_{j=1}^{M} \int_{R^3\times\mathcal{M}(3)\times\mathcal{A}} q_S^b\Big(\vec{x}_t(M), \vec{x}_t(M)\oplus y^{(j)}\Big)dy \\
&+\sum_{j=1}^{M} q_S^d\Big(\vec{x}_t(M), \wp_S^{(j)}\vec{x}_t(M)\Big).
\end{aligned}
\tag{4.2}
$$

The birth/death intensities $q_T^b, q_T^d, q_S^b, q_S^d$ can be derived from the posterior measures of the present and the candidate configurations in two ways. One is analogous to a Gibbs sampling type algorithm while the other is analogous to a Metropolis type acceptance/rejection algorithm. The intensities obtained from the first method are given by,

- Gibb's sampling:

$$
\begin{aligned}
q_T^b(\vec{x}_t(M), \vec{x}_t(M)\oplus_j \vec{y}_t(1)) &= \pi(\vec{x}_t(M)\oplus_j \vec{y}_t(1)) \\
q_S^b(\vec{x}_t(M), \vec{x}_t(M)\oplus_j y^{(j)}) &= \pi(\vec{x}_t(M)\oplus_j y^{(j)})) \\
q_T^d(\vec{x}_t(M), \wp_T^{(j)}(\vec{x}_t(M))) &= \pi(\wp_T^{(j)}(\vec{x}_t(M))) \\
q_S^d(\vec{x}_t(M), \wp_S^{(j)}(\vec{x}_t(M))) &= \pi(\wp_S^{(j)}(\vec{x}_t(M)))
\end{aligned}
$$

These expressions provide the birth/death intensities for constructing the jump process having jump moves derived from Gibb's sampling. Our implementation is based on the jump moves of second type which represents a modification of the Metropolis algorithm introduced in 1953 [33]. This algorithm is implemented through the following steps.

Metropolis Based Jump-Algorithm

1. Generate independent exponential r.v.'s $u_1, u_2, ..$ with the intensity λ, where λ is the average number of diffusion cycles for every jump move.

2. At time $t_i = \sum_{j=1}^{i} u_j$ draw a candidate $\vec{y}_t(M')$ from the prior (e.g by using the equations of motion).

3. Compute $L_t(\vec{y}_t(M'))$.

    If $[L_t(\vec{x}_t(M)) - L_t(\vec{y}_t(M'))] > 0$,

        go to $\vec{y}_t(M')$.

    else

        go to $\vec{y}_t(M')$ with the probability $e^{-[L_t(\vec{y}_t(M'))-L_t(\vec{x}_t(M))]}$.

4. Repeat step 2.

The corresponding birth/death intensities for $\vec{y}_t(M') \in \mathcal{X}(\mathcal{T}^1(\vec{x}_t(M)))$ are given by

$$\begin{aligned}
q_T^b(\vec{x}_t(M), \vec{x}_t(M) \oplus_j \vec{y}_t(1)) &= \frac{1}{4(M+1)} e^{-[L(\vec{x}_t(M)\oplus_j \vec{y}_t(1))-L(\vec{x}_t(M))]_+} \frac{e^{-P(\vec{y}_t(1))}}{Z_T(1)}, \\
q_S^b(\vec{x}_t(M), \vec{x}_t(M) \oplus_j y^{(j)}) &= \frac{1}{4M} e^{-[L(\vec{x}_t(M)\oplus_j y^{(j)})-L(\vec{x}_t(M))]_+} \frac{e^{-P(y^{(j)})}}{Z_S(1)}, \\
q_T^d(\vec{x}_t(M), \wp_S^{(j)} \vec{x}_t(M)) &= \frac{1}{4M} \frac{e^{-[L(\wp_S^{(j)} \vec{x}_t(M))-L(\vec{x}_t(M))]_+}}{Z_T(1)}, \\
q_S^d(\vec{x}_t(M), \wp_S^{(j)} \vec{x}_t(M)) &= \frac{1}{4M} \frac{e^{-[L(\wp_S^{(j)} \vec{x}_t(M))-L(\vec{x}_t(M))]_+}}{Z_S(1)}, \qquad (4.3)
\end{aligned}$$

where $Z_T(1), Z_S(1)$ are the partition functions for the prior densities on single track and single track-segment configurations respectively. In between jump moves the process stays in the current subspace and performs stochastic gradient search via diffusion process.

4.2.2 Diffusion Process

The diffusion process contributes in the search of the features which lie in the continuous space. It is a sample path continuous process which performs a randomized gradient search on the posterior potential $E_t(\vec{x}_t(M))$ in the current subspace $\mathcal{X}_t(M)$ according to Langevin's stochastic differential equation (SDE).

A diffusion is completely defined by its infinitesimal mean and variance (Appendix 3). In this approach we generate a diffusion by assigning the gradients of the posterior energy to be the infinitesimal mean. For the sub-space containing M tracks the diffusion flows through the manifold $\mathcal{X}_t(M)$ estimating the continuous parameters: the target's positions $\vec{p} \in \Re^3$ and orientations $\vec{\phi} \in \mathcal{M}(3)$. Define $n_M = \sum_{m=1}^{M} t^{(m)}$, the total track segments in $\vec{x}_t(M)$ and associate with the first $3n_M$ components of the S.D.E. $X(s)$ the flow through $\mathcal{M}(3)^{n_M}$ to estimate the orientations as described in [34], and the last $3n_M$ components the flow through $\Re^{(3n_M)}$ to estimate the positions. Then the diffusion $X(s)$ satisfies the following vector S.D.E.:

$$\begin{aligned} X_1(s) &= \left[X_1(0) + \int_0^s -\frac{1}{2}\nabla_1 E_t(X(\tau))d\tau + W_1(s)\right]_{\text{mod } 2\pi}, \qquad (4.4) \\ X_2(s) &= X_2(0) + \int_0^s -\frac{1}{2}\nabla_2 E_t(X(\tau))d\tau + W_2(s)\ , \qquad (4.5) \end{aligned}$$

where $[\cdot]_{\text{mod } 2\pi}$ is taken componentwise, $(X(s) = [X_1(s), X_2(s)])$, and $W_1(s), W_2(s)$ are the standard vector Wiener processes of dimensions $3n_M$, and ∇_1, ∇_2 are the gradients with respect to the vectors $\vec{p}_t(M), \vec{\phi}_t(M)$.

4.2.3 Ergodic Result

Now we present an important result on the ergodic properties of jump-diffusion process. This result verifies the claim that the jump-diffusion process constructed above samples from the posterior distribution $\pi_t(\vec{x}_t(M))$.

Theorem 1 *If the jump diffusion process $X(s)$ has the properties that:*

(a) the diffusion $X(s)$ within any subspace satisfies the S.D.E. of Eqns (4.4,4.5)

(b) the birth/death intensities defined by Eqn (4.3) generate the jump process with the parameters given by Eqns (4.1,4.2)

then $X(n\Delta)$ converges in variation norm to $\mu_t(d\vec{x}_t(M)) = \pi_t(\vec{x}_t(M))d\vec{x}_t(M)$.

Proof: See Appendix 4.

In Eqn (4.5), the term $\nabla_2 E_t(X(s))$ has two gradient components: the likelihood gradient $\nabla_2 L_t(X(s))$ and the prior gradient $\nabla_2 P(X(s))$. Since we use a Gaussian prior on target positions the gradient of prior potential is given by $K_p^{-1}\vec{p}_t(M)$. Clearly, for the dynamic scenarios with changing configurations and therefore changing covariances the matrix-inverse computation gets intensive. We now present an alternate diffusion process which also samples from the same density and is computationally far less intensive. This diffusion follows the SDE,

$$X_2(s) = X_2(0) + \int_0^s -\frac{1}{2}\left(K_p \nabla_2 L_t(X(\tau)) + X(\tau)\right) d\tau + \sqrt{K_p} dW_2(s) \qquad (4.6)$$

where K_p is the $3n_M \times 3n_M$ covariance matrix of the position vector $\vec{p}_t(M)$ and $W_2(s)$ is the standard Wiener process of dimension $3n_M$. The following theorem concludes that this diffusion also samples from the same posterior distribution.

Corollary 1 *The modified jump-diffusion process with the properties that:*

(a) the diffusion $X(s)$ within any subspace satisfies the S.D.E. of Eqns (4.4,4.6),

(b) the birth/death intensities defined by Eqn (4.3) generate the jump process with the parameters given by Eqns (4.1,4.2),

then $X(n\Delta)$ converges in variation norm to $\mu_t(d\vec{x}_t(M)) = \pi_t(\vec{x}_t(M))d\vec{x}_t(M)$.

Proof: See Appendix 5.

5. Implementation and Results

Now we present the implementation of a jump-diffusion algorithm for estimating the motion of a single target, i.e. M = 1. The algorithm was jointly implemented using a Silicon Graphics workstation for data generation and visualization, and a massively parallel 4096 processor SIMD DECmpp machine for implementing the track-recognition algorithm. The recognition component of the algorithm was implemented by my fellow student, Robert S. Teichman, who also contributed in joining the tracking and recognition components together.

The parameter set describing the target configuration for the observation interval $\{1, ..t\}$, is

$$\begin{aligned} \vec{x}_t(1) &= \{\vec{x}^{(1)}(k) : k \in \mathcal{T}^{(1)}\} \\ &\in \mathcal{X}_t(1) \equiv \bigcup_{t^{(1)}=1}^{t} \left(\mathcal{M}(3) \times \Re^3 \times \mathcal{A}\right)^{t^{(1)}} \times \aleph \end{aligned}$$

For estimating the object orientation, the orientation space $\mathcal{M}(3)$ was uniformly sampled and 64 × 64 pixel 2D projections $\mathcal{P}(\cdot)$ of the 3D surface of the target at sampled orientations were generated and stored. This set of templates form the object space over which the recognition is performed, i.e the estimates are selected from this set.

5.1 Data Simulation

The *flight simulator* software on Silicon Graphics workstation was utilized to generate parameterized airplane paths. These path coordinates were then used in data simulation modules on mpp and Silicon Graphics to obtain data sets corresponding to the narrowband tracking array and the optical imaging sensor, respectively. The tracking data consists of a 64 length complex vector sampled at the cross-array $\vec{y}_1(k)$ and a real number corresponding to the range data for the target $r(k)$ at each time index k. The imaging data, generated by optical imaging of the space around the estimated position of the target using Silicon Graphics, consists of a 64×64 matrix $\vec{y}_2(k)$ of grey scale pixel values for each observation. The data set is then transferred to the mpp where the orientation and target type estimation is performed. In the results presented we have used high signal to noise ratio for both the tracking and imaging data sets. The noise in the data model was generated using the Gaussian random number generator on the machines.

5.2 Estimation Algorithm

Now we describe the jump-diffusion algorithm for estimating the single track scene. The estimation algorithm proceeds by births and deaths of track segments at random times through discrete jump moves. At any given time t the jump-diffusion algorithm is run to generate samples from the posterior distribution given by $\pi_t(\cdot)$. This simulation is performed till the next data set arrives when the algorithm starts sampling from $\pi_{t+1}(\cdot)$ and so on. The two possible jump moves involve the following transformations in the parameter space,

$$\begin{aligned}\vec{x}_t(1) &\rightarrow \vec{x}_t(1) \oplus_1 y^{(1)} \in \left(\Re^3 \times \mathcal{M}(3) \times \mathcal{A}\right)^{t^{(1)}+1} \times \aleph \,, \\ \vec{x}_t(1) &\rightarrow \wp_S^{(1)} \vec{x}_t(1) \in \left(\Re^3 \times \mathcal{M}(3) \times \mathcal{A}\right)^{t^{(1)}-1} \times \aleph \,.\end{aligned}$$

where $\vec{x}_t(1) \in (\Re^3 \times \mathcal{M}(3) \times \mathcal{A})^{t^{(1)}} \times \aleph$. We assign equal probabilities for selecting one of the two options with the actual moves being performed on the basis of posterior energies. Notice that for the option to delete the track segment there is only one candidate configuration but for adding the track segment the candidates are numerous corresponding to all possible values of $y^{(1)}$. In that case, using Metropolis based jump moves the algorithm candidates are generated and selected as follows. The vector differential equation (Eqn 3.3), describing the motion in the body frame coordinates, can be written in discrete form as

$$\vec{v}(k+1) = (A(k) - I_3)\vec{v}(k) + \vec{f}(k) \,.$$

Suppose the track is estimated up to the $(k+1)^{st}$ stage and the $k+2$ position (or equivalently the $k+1$st velocity component) is to be found. The estimated velocity profile for times $1, 2, .., k$ is available. A sample from $N(0, \sigma_0^2 I_3)$ is substituted for the force vector and the difference equation is solved for $\vec{v}(k+1)$ using the estimated rotational motion and the previous velocity estimate. Using the velocity-position transformation (Eqn 3.1) this $\vec{v}(k+1)$ provides a candidate for the inertial position $\vec{p}(k+2)$. The orientation and target type components for the candidate segment are chosen to be the same as the previous segment. This new track segment is selected and added to the track estimate according to the Metropolis type jump algorithm (discussed in chapter 4). The likelihood potential of this candidate, $L(\vec{x}_t(1) \oplus_1 y^{(1)})$, is compared to the likelihood potential of the present estimate, $L(\vec{x}_t(1)$. If $L(\vec{x}_t(1) > L(\vec{x}_t(1) \oplus_1 y^{(1)})$, then the segment $y^{(1)}$ is added to the track $\vec{x}_t(1)$ otherwise it is added with the probability $e^{-(L(\vec{x}_t(1)\oplus_1 y^{(1)})-L(\vec{x}_t(1)))}$.

Then the algorithm adjusts the positions and orientations in the estimated track following the gradients of the posterior according to the diffusion equations until the next jump move is performed. The average number of diffusion cycles per jump move is given by the parameter λ, the mean of exponential times separating jump moves. It should be noted that the gradients of imaging data likelihood involves the derivative of the projection transform which cannot be derived analytically. These are approximated numerically at the sample orientations by taking the difference of adjacent pre-stored templates at that orientations, scaled by the step size of parameter variation. The gradients of the tracking data likelihood are derived analytically using the chain rule for coordinate transformations. The diffusion process is simulated via the discrete equations corresponding to the SDE's 4.4,4.6 given by

$$\begin{aligned} X_1((n+1)\epsilon) &= [X_1(n\epsilon) - \frac{1}{2}\nabla_1 E_t(X(n\epsilon)) + [W_1((n+1)\epsilon) - W_1(n\epsilon)]]_{mod\ 2\pi} \\ X_2((n+1)\epsilon) &= X_2(n\epsilon) - \frac{1}{2}\nabla_2 E_t(X(n\epsilon)) + [W_2((n+1)\epsilon) - W_2(n\epsilon)] \end{aligned}$$

Figure 5.1 shows three successive stages of the algorithm for estimating a portion of the true track shown in grey with the estimates overlapping in white. The algorithm proceeds via sequence of jump moves corresponding to the births of track-segments and adjustment of the track-estimates between the jumps via the diffusion algorithm.

Shown in Fig 5.2 is the result for the complete track estimation algorithm. The left panel shows the simulation environment for the implementation. The mesh represents the ground supporting the inertial frame of reference and the sensor systems while the grey track represents the parameterized plane path generated from the flight simulator on Silicon Graphics and used as true track in the simulations. In the right panel the estimated track is shown in white overlapping the true track obtained via the jump-diffusion algorithm.

5.2.1 Parallel Processing

The DECmpp has a 64 × 64 mesh of processors each of which can simultaneously operate on a matrix of up to 64 × 64 data elements. In this application a large part of the computation involves simultaneous operations like coordinate transformations, trigonometric operations, and matrix computations on arrays of data. These operations along with global summation and processor communication are efficiently implemented on a machine like the DECmpp. The choice of sizes of 64-length tracking data vector and 64 × 64 imaging data allows convenient mapping of the problem onto the 64 × 64 processor array of mpp.

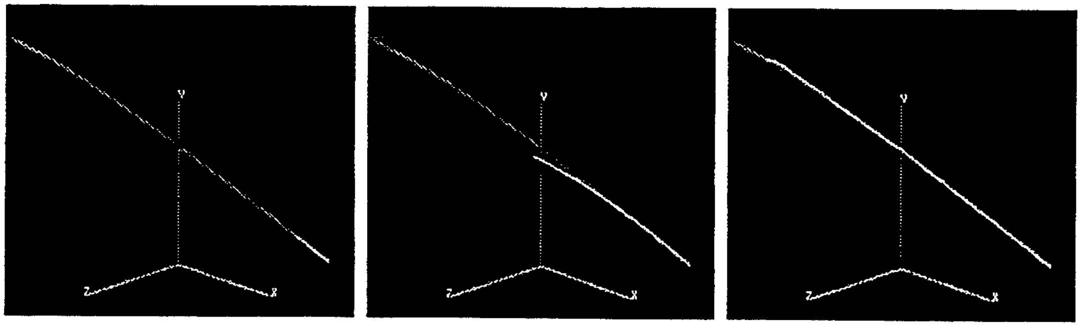


Figure 5.1: Jump-diffusion estimation of a portion of the track. The true track is drawn in grey while the estimates overlap in white. The estimation proceeds via a sequence of jump moves with the diffusion cycles performed between moves.

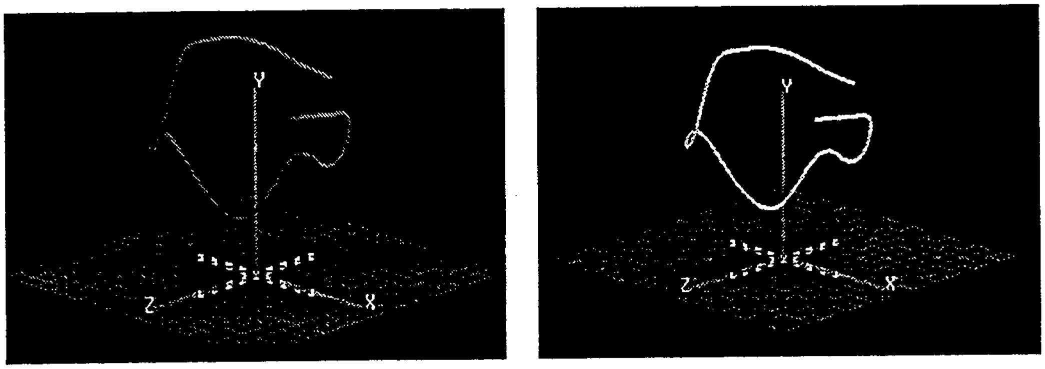


Figure 5.2: 3D track estimation: The left panel shows the actual track drawn in gray with the mesh representing ground supporting the observation system in the inertial frame of reference. The right panel displays the results from the single track estimation with the estimates drawn in white.

5.2.2 Remote Visualization

Even though the massively parallel machine is ideal for implementing estimation algorithm it doesn't have adequate graphical resources to provide good display of results. The Silicon Graphics workstation is well suited for 3-D visualization of the actual flight path and the estimated path. Therefore we distribute the tasks across various platforms to make use of advance computing and graphical resources that are not available on any one machine. In fact there could be multiple visualization nodes to address various aspects of the implementation. Also the distributed computation implies more efficient implementation as the tasks are shared by various machines.

The communication between machines demands pipeline of high speed network for data flow. This was implemented using TCP/IP sockets on ethernet to which the mpp and SGI machines were connected. The algorithm performs the computation continuously while feeding the estimates onto the network at regular intervals. The estimates are received by SGI and fed into a visualization program to display the results in a desired way.

6. Tracking the Direction of Arrivals of Multiple Moving Targets

This problem involves only the object tracking in a familiar 'direction of arrival' set-up. The goal is to track the motion of an unknown number of signal sources assumed to be present in the plane (2-D space) containing the uniform linear array of sensors shown in Figure 6.1. Here we are interested in only one angular location $\theta \in [0, \pi]$ (azimuth or elevation) of the objects. Even though the problem is posed for $\theta \in [0, \pi]$, we define the parameter space to be $[0, 2\pi]$ in order to utilize results from [34] to prove the theorem supporting the algorithm. So far these results have been derived for the parameter spaces which form Lie groups. Hence, we will treat the parameter space to be $\theta \in \mathcal{M}(1)$ where $\mathcal{M}(1) = [0, 2\pi]$ with 0 , 2π identified

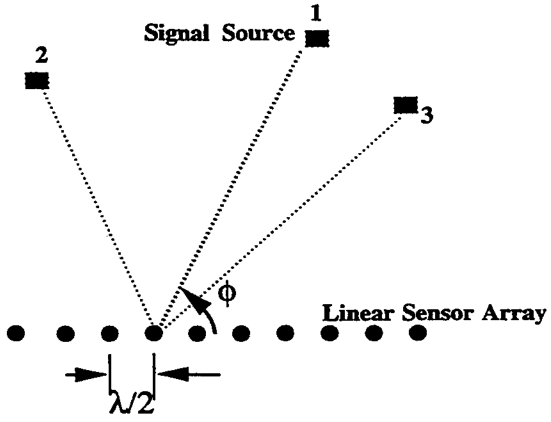


Figure 6.1: The linear array of uniformly placed isotropic sensors to track the direction of arrival of multiple signal sources.

which is a Lie group. In this chapter the notation is similar except that the vectors are denoted by $\vec{c}_t(\cdot)$ and the parameter spaces by $\mathcal{C}_t(\cdot)$. For m^{th} target the parameter configuration, a sequence of its locations during its stay in the scene given by $\mathcal{T}^{(m)} = \{t_1^{(m)}+1, .., t_1^{(m)}+t^{(m)}\}$, is

$$\{\theta^{(m)}(k) : k \in \mathcal{T}^{(m)}\} \in \mathcal{C}_t(1) \equiv \bigcup_{t^{(m)}=1}^{t} \mathcal{M}(1)^{t^{(m)}} \times \aleph ,$$

and for a multiple target scenario with M objects the configuration of significant parameters is

$$\vec{c}_t(M) = \bigcup_{m=1}^{M} \{\theta^{(m)}(k) : k \in \mathcal{T}^{(m)}\} \in \mathcal{C}_t(M) \equiv \prod_{m=1}^{M} \left(\bigcup_{t^{(m)}=1}^{t} \mathcal{M}(1)^{t^{(m)}} \times \aleph \right) .$$

Since M, the number of targets present, is unknown the complete parameter space is defined to be the countable, infinite union of subspaces each having an associated value of M, i.e. $\mathcal{C}_t = \cup_{M=1}^{\infty}\mathcal{C}_t(M)$. So the aim is to estimate $\vec{c}_t(M) \in \mathcal{C}_t$ given the observed data set using minimum-mean-square-error (*MMSE*) approach.

6.1 Bayesian Posterior

6.1.1 Data Likelihood

The model used describes the multi-track, multi-sensor scenario in the following way. Consider the signal from m^{th} source arriving at angle $\theta^{(m)}(k)$ with amplitude $s^{(m)}(k)$ at the sensor array. Define the response at the sensor array at discrete time k as $\vec{y}(k)$ made up of the superposition of the incoming signal with ambient noise and is given by

$$\vec{y}(k) = \vec{d}(\theta^{(m)}(k))s^{(m)}(k) + \vec{n}_3(k), \tag{6.1}$$

where $\vec{d}(\theta^{(m)}(k))$ is the 10 × 1 array of delays and gains corresponding to the 10-element linear array and $\vec{n}_3(k)$ is a 10 × 1, 0-mean complex Gaussian random vector with covariance $\sigma_3^2\mathbf{I}$. For the cases studied here there are M signals incident upon the array, with $\{\theta^{(m)}(k), s^{(m)}(k)\}_{m=1}^{M}$,M *unknown* integer. The measured signal $\vec{y}(k)$ for a multiple target scene is modeled as

$$\vec{y}(k) = \sum_{m=1}^{M} \vec{d}(\theta^{(m)}(k))1_k(\mathcal{T}^{(m)})s^{(m)}(k) + \vec{n}_3(k). \tag{6.2}$$

Each of $k = 1$,2...t time samples of data are received with perhaps a different mean corresponding to the motion of targets.

We assume that $s^{(m)}(k)$ are unknown, deterministic quantities which enter into the *mean* of the observation $\vec{y}(k)$ as $\sum_{m=1}^{M} \vec{d}(\theta^{(m)}(k))1_k(\mathcal{T}^{(m)})s^{(m)}(k)$, for the simplicity of presentation $s^{(m)}(k)$ are assumed to be known. The likelihood energy associated with the data collected up to time t becomes

$$L_t(\vec{c}_t(M)) = -\frac{1}{2\sigma_3^2}\sum_{k=1}^{t}\left|\vec{y}(k) - \sum_{m=1}^{M} \vec{d}(\theta^{(m)}(k))1_k(\mathcal{T}^{(m)})s^{(m)}(k)\right|^2 . \tag{6.3}$$

6.1.2 von-Mises Prior

The Bayes posterior $\pi_t(\vec{c}_t(M))$ at time t for an M-track scene can be constructed as follows. Define the von-Mises prior [24] for the M-track scene to have the Gibbs potential

$$P_t(\vec{c}_t(M)) = \kappa' \sum_{m=1}^{M} \sum_{k\in\mathcal{T}^{(m)}} cos(\theta^{(m)}(k) - \theta^{(m)}(k-1)) . \tag{6.4}$$

Notice, for $\kappa' > 0$ a large positive number the prior favors smooth tracks.

Then the posterior energy becomes

$$E_t(\vec{c}_t(M)) = \sum_{m=1}^{M} \sum_{k\in\mathcal{T}^{(m)}} \kappa'(cos(\theta^{(m)}(k) - \theta^{(m)}(k-1))) \tag{6.5}$$

$$-\frac{1}{\sigma_3^2}\sum_{k=1}^{t} \left|\vec{y}(k) - \sum_{m=1}^{M} \vec{d}(\theta^{(m)}(k))1_k(\mathcal{T}^{(m)})s^{(m)}(k)\right|^2 .$$

This posterior energy is conditoned on a given M, the complete posterior is the convex combination of the energies for all values of M as in the previous problem.

6.2 Sampling the Multi-Track Space

Now we define the algorithm for sampling the posterior $\pi_t(\vec{c}_t(M))$ conditioned on the data up to time t over the multi-track space $\mathcal{C}_t$. We construct a Markov *jump-diffusion* process $C(s) \in \mathcal{C}_t$ which visits the candidate solutions according to the posterior density $\pi_t(\vec{c}_t(M)) = Z^{-1}e^{-E_t(\vec{c}_t(M))}$. This Markov process is said to satisfy *jump-diffusion* dynamics in the sense that (i) on random exponential times the process jumps from one of the countably infinite non-connected set of spaces $\mathcal{C}_t(M)$ to another, and (ii) between jumps it performs diffusion cycles generated by the S.D.E.'s appropriate for that space.

6.2.1 Diffusion S.D.E.'s

The diffusion part of the process essentially follows stochastic gradient descent in each of the subspaces $\mathcal{C}_t(M)$ on the posterior potential $E_t(\vec{c}_t(M))$ by following a

Langevin's stochastic differential equation (S.D.E.) appropriate for the current subspace. For the sub-space containing M tracks the diffusion will flow through the manifold $\mathcal{C}_t(M)$. Then the diffusion $C(s)$ satisfies the following vector S.D.E.:

$$C(s) = \left[C(0) + \int_0^t -\frac{1}{2} \nabla_{(n_M)} E_t(C(\tau)) d\tau + W(s) \right]_{\mathbf{mod}\ 2\pi} \tag{6.6}$$

where $[\cdot]_{\mathbf{mod}\ 2\pi}$ is taken componentwise, $W(s)$ is the standard vector Wiener processes belonging to the space of dimension $\sum_{m=1}^{M} t^{(m)}$ and $\nabla E_t(C(\dot{\tau}))$ is the gradient with respect to the set $\vec{c}_t(M)$.

6.2.2 Jump process

The jump process carries the inferences from subspace to subspace via the simple jump moves based on the classical ideas of birth and death processes. The jump intensities of these simple moves are governed by the posterior potential of the candidate configurations. These jumps accommodate the following kinds of model order changes: (i) those corresponding to the addition and deletion of track segments to and from existing tracks, and (ii) the addition or deletion of new or existing tracks. Notationally, these jump moves are represented by

$$\begin{aligned}
\text{Addition of } j\text{th track } &: \vec{c}_t(M)) &\rightarrow&\quad \vec{c}_t(M) \oplus_j \vec{c'}_t(1)\ , \\
\text{Addition of segment to } j\text{th track } &: \vec{c}_t(M) &\rightarrow&\quad \vec{c}_t(M) \oplus_j (\theta')^{(j)} \\
\text{Deletion of } j\text{th track } &: \vec{c}_t(M) &\rightarrow&\quad \wp_T^{(j)} \vec{c}_t(M)\ , \\
\text{Deletion of a segment from } j\text{th track } &: \vec{c}_t(M) &\rightarrow&\quad \wp_S^{(j)} \vec{c}_t(M).
\end{aligned}$$

Define $\mathcal{T}^1(\vec{c}_t(M))$ to be the set of configuration types reachable from $\vec{c}_t(M)$ in one jump move, i.e.

$$\mathcal{T}^1(\vec{c}_t(M)) = \{\vec{c}_t(M) \oplus_j \vec{c'}_t(1), \vec{c}_t(M) \oplus_j (\theta')^{(j)}, \wp_T^{(j)}\vec{c}_t(M), \wp_S^{(j)}\vec{c}_t(M)\}$$

and $\mathcal{C}_t(\mathcal{T}^1(\vec{c}_t(M)))$ be the space containing these configurations types. The transition parameters for these moves are given by

$$\begin{aligned}
q(\vec{c}_t(M), d\vec{c'}_t(M+1)) &= \sum_{j=1}^{M+1} q_T^b(\vec{c}_t(M), \vec{c'}_t(M+1))d(\vec{c'}_t(1))\delta_{\vec{c}_t(M)}\Big(d(\wp_T^{(j)}\vec{c'}_t(M+1))\Big) , \\
q(\vec{c}_t(M), d\vec{c'}_t(M)) &= \sum_{j=1}^{M} q_S^b(\vec{c}_t(M), \vec{c'}_t(M)d(\theta)\delta_{\vec{c}_t(M)}\Big(d\Big(\wp_S^{(j)}\vec{c'}_t(M)\Big)\Big) , \\
&+ \sum_{j=1}^{M} q_S^d(\vec{c}_t(M), \vec{c'}_t(M))\delta_{\wp_S^{(j)}\vec{c}_t(M)}\Big(d\vec{c'}_t(M)\Big) , \\
q(\vec{c}_t(M), d\vec{c}_t(M-1)) &= \sum_{j=1}^{M} q_T^d(\vec{c}_t(M), \vec{c'}_t(M-1))\delta_{\wp_T^{(j)}\vec{c}_t(M)}\Big(d\vec{c'}_t(M-1)\Big) .
\end{aligned}$$

The intensity for jumping out of configuration $\vec{c}_t(M)$ is given by

$$\begin{aligned}
q(\vec{c}_t(M)) &= \int_{\mathcal{C}_t(\mathcal{T}^1(\vec{c}_t(M)))} q(\vec{c}_t(M), d\vec{c'}_t(M')) && (6.7) \\
&= \sum_{j=1}^{M+1} \int_{\mathcal{M}(1)\times\aleph} q_T^b\Big(\vec{c}_t(M), \vec{c}_t(M) \oplus_j \vec{c'}_t(1)\Big)d\vec{c'}_t(1) \\
&+ \sum_{j=1}^{M} q_S^d\Big(\vec{c}_t(M), \wp_S^{(j)}\vec{c}_t(M)\Big) \\
&+ \sum_{j=1}^{M} \int_{\mathcal{M}(1)} q_S^b\Big(\vec{c}_t(M), \vec{c}_t(M) \oplus_j \theta^{(j)}\Big)d\theta \\
&+ \sum_{j=1}^{M} q_T^d\Big(\vec{c}_t(M), \wp_T^{(j)}\vec{c}_t(M)\Big) , && (6.8)
\end{aligned}$$

and the transition measure given by

$$Q(\vec{c}_t(M), d\vec{c}_t(M')) = \frac{q(\vec{c}_t(M), d\vec{c}_t(M'))}{\int_{C_t(T^1(\vec{c}_t(M)))} q(\vec{c}_t(M), d\vec{c}_t(M'))} \, .$$

As mentioned earlier the choice of the intensities $q_T^d, q_T^b, q_S^b, q_S^d$ is not fixed. Here we use an algorithm analogous to the Gibb's sampling to perform the jump moves described as follows: Generate a sequence of independent samples $\Delta_0, \Delta_1, \ldots$ from the exponential distribution with parameter λ, then starting at $C(0) \in \mathcal{C}_t(M)$ run the S.D.E. given in equation (6.6) until time τ_0 such that $\int_0^{\tau_0} q(C_M(s))ds = \Delta_0$. Then carry out a jump according to $Q(C(\tau_0), dc')$ to some point $c_1 \in \mathcal{C}_t(M')$, so that $C(\tau_0) = c_1$. Run the SDE on $\mathcal{C}_t(M')$ starting at c_1 until time τ_1 such that $\int_0^{\tau_1} q(C_{M'}(s))ds = \Delta_1$ and so on. The associated birth/death intensities $q_T^b, q_T^d, q_S^b, q_S^d$ are presented via a theorem in next section.

6.2.3 Ergodic Theorem

We now state our theorem on the ergodic properties of the jump-diffusion process.

Theorem 2 *If the jump diffusion process $C(s)$ has the properties that:*

(a) the diffusion $C(s)$ within any subspace satisfies the S.D.E. of Eqns (6.6),

(b) the birth/death intensities

$$\begin{aligned}
q_T^b(\vec{c}_t(M), \vec{c}_t(M) \oplus_j \vec{c}_t(1)) &= \pi(\vec{c}_t(M) \oplus_j \vec{c}_t(1)) \, , \\
l_S^b(\vec{c}_t(M), \vec{c}_t(M) \oplus_j \theta^{(j)} &= \pi(\vec{c}_t(M) \oplus_j \theta^{(j)}) \, , \\
q_T^d(\vec{c}_t(M), \wp_T^{(j)} \vec{c}_t(M) &= \pi(\wp_T^{(j)} \vec{c}_t(M)) \, , \\
q_S^d(\vec{c}_t(M), \wp_S^{(j)} \vec{c}_t(M) &= \pi(\wp_S^{(j)} \vec{c}_t(n_M, M)) \, ,
\end{aligned}$$

form a jump process with the parameters given by the Eqns 6.7,6.8, then $C(n\Delta)$ *converges in variation norm to* $\mu_t(d\vec{c}_t(M)) = \pi(\vec{c}_t(M))d\vec{c}_t(M)$.

Proof: See Appendix 6.

Remark: As proven in Amit [34] Eqn (6.6) is simulated via the discrete equation

$$C((k+1)\epsilon) = \left[C(k\epsilon) - \frac{1}{2}\nabla E_t(C(k\epsilon)) + [W((k+1)\epsilon) - W(k\epsilon)]\right]_{\mathrm{mod}(2\pi)}, \quad (6.9)$$

which provides a sequence of pathwise convergent approximations to the diffusion $C(t)$ on the Torus as $\epsilon \to 0$.

6.3 Implementation and Results

The algorithm was implemented on massively parallel AMT-DAP 510 SIMD machine. It has a 32 × 32 mesh of processors suitable for the parallel computations involving up to 1024 data elements. In this implementation the data sets were simulated for sensor array having 10 elements at half wavelength spacing observing a variable number of tracks length 10 each. Also it is assumed that all targets are present at all times.

The estimation proceeds by simple jump moves at random times with diffusion cycles performed in between jump moves. It estimates the number of tracks by birth/death of tracks and estimates the track lengths by birth/death of track segments. At any given time t the jump-diffusion algorithm is run to generate samples from the posterior distribution given by $\pi_t(\vec{c}_t(n_M, M))$. This sampling goes on till the next data sample time when the algorithm starts sampling from $\pi_{t+1}(\vec{c}_t(n_M, M))$. There are four kinds of jump moves possible at all jump times τ_i: addition/deletion of a track-segment and addition/deletion of a track. The algorithm chooses one of them with equal probability and performs candidate generation and selection according to the Gibb's sampling. Shown in Figure (6.3) are two kinds of jump moves being performed by the algorithm. The vertical axis represents time variable while the horizontal axis parameterizes the angular location of targets. Shown in the background is the spatial power spectrum, generated from the data using *minimum variance distortionless response* (MVDR) beamforming [35] with brightness representing spatial power. Superimposed on it are the actual tracks in grey while the estimated tracks are drawn in white. The two panels show the estimation results at two different stages. The algorithm carries the configuration in the left panel to the one in right by birth of a track and by birth of a track segment at the end of existing track.

Shown in Figure 6.3 are the implementation results of a jump-diffusion algorithm estimating a four-track configuration observed via the linear sensor array.

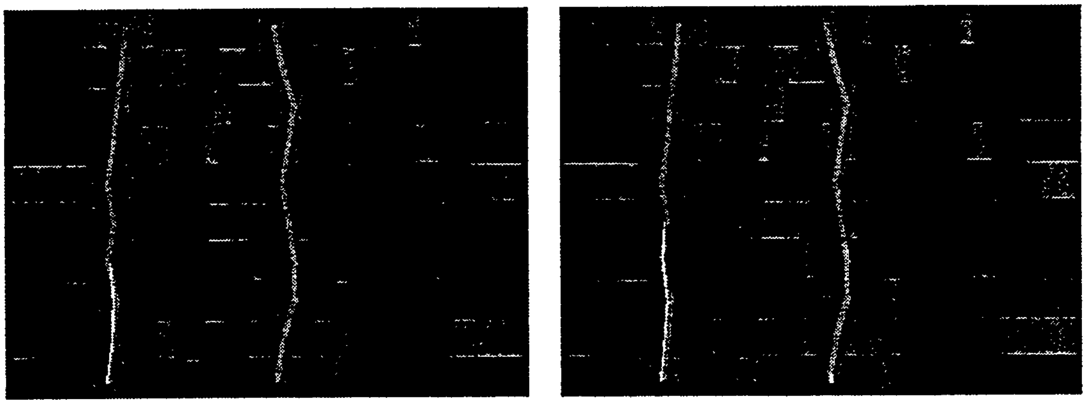


Figure 6.2: The two kinds of jump moves: addition of a track and addition of a track-segment at the end of existing track estimate. The actual tracks are shown in grey with the estimates overlapping in white. Underlying spatial power spectrum generated via MVDR method shows the noisy data.

The algorithm does not know apriori the number of tracks in the scene. The four panels show successive stages of estimation algorithm as the jump-diffusion process proceeds. The algorithm was run with the jump process adding tracks and track-segments, and the diffusion part constantly adjusting the estimated parameters. The lower left panel shows the rough initial track estimates improving via the continuous diffusion deformations with the result shown in the lower right panel.

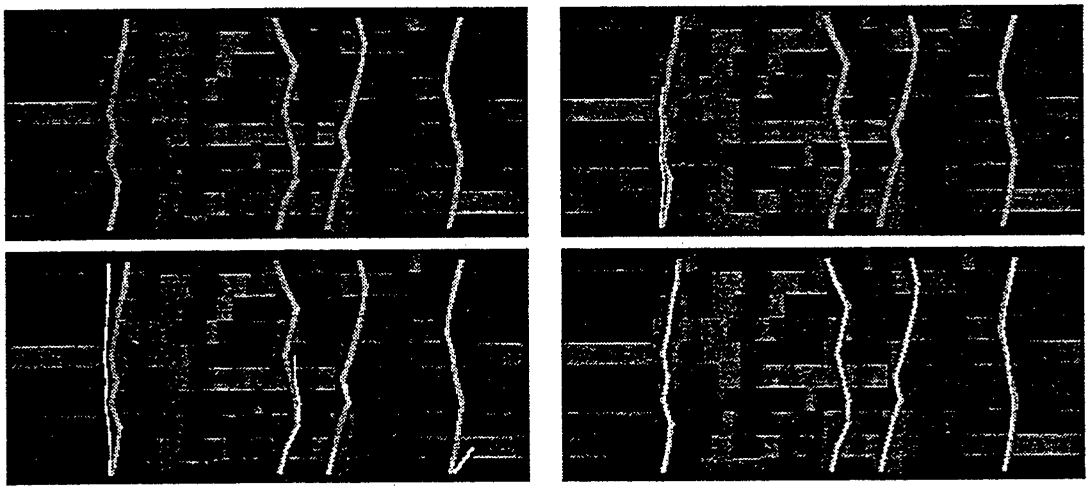


Figure 6.3: Estimation of a four-track scene: the upper left panel shows the true tracks in grey with the spatial power spectrum generated via MVDR from the noisy data. The other three panels show the successive stages of the algorithm with the final result shown in the lower right panel.

7. Future Work

In this chapter we discuss the future work which could be performed in this field to augment the present results. There are many areas of interest which could be explored further.

As mentioned in chapter 3 that there is a set of differential equations governing the rotational and translational motions of the moving airplane. At present we utilize a subset of them to impose a dynamics based prior on the sequence of airplane positions. The extension of similar analysis on rest of the equations can provide a prior for the rotational parameters. We propose to utilize the last three equations of motion listed in chapter 3 for deriving a dynamics based prior on the rotational parameters.

We are also interested in exploring the alternate representations of the parameter space. The torus representing the parameter space of the Euler's angles also forms a special orthogonal group SO(n) in a higher dimensional Euclidean space. Therefore the SO(n) structure can be utilized in this representation and corresponding analysis can be followed. There is a rich treatment in the literature on the algebraic analysis of special orthogonal groups.

Automated target tracking and recognition fall under the category of sequential estimation problems. The estimation is performed conditioned on the increasing family of σ-algebras generated by the observations. Consider $\mathcal{Y}_t$ to be the σ-algebras associated with the data collected till time t. The MMSE estimate of the parameters $\vec{x}_t$ is given by the conditional mean $\hat{\vec{x}}_t = \mathcal{E}(\vec{x}|\mathcal{Y}_t)$. We wish to explore the limiting behavior of $\lim_{t\to\infty} \hat{\vec{x}}_t$ in comparison with $\mathcal{E}(\vec{x}|\mathcal{Y}_\infty)$ and are interested in the asymptotic consistency of the estimates.

8. ACKNOWLEDGMENTS

I owe great gratitude to my parents for their enormous contribution in shaping up my educational career and their invaluable moral support at every stage.

I feel grateful to my academic advisor, Dr Michael I. Miller, for the inspiration, encouragement and guidance I received from him for this work and elsewhere.

Thanks to Dr. Donald L. Snyder, Dr. Joseph A. O'Sullivan and Dr. Daniel R. Fuhrmann for their help in my learning and their contribution in the work presented here. I also thank them for being my thesis committee members.

Additional thanks go to my fellow graduate students and friends who shared my life on both the good and bad days.

This research was supported by ONR N00014-92-J-1418, ARO DAA 03-92-6-0141, and Rome Laboratory F30602-92-C-004.

9. Appendix

Appendix 9.1 Cross-Array Manifold

The signal received from the target located in 3-D observation space is sampled by the cross-array of sensors placed as shown in the figure 3.2. This cross-array consists of two uniform linear arrays orthogonal to each other, sensitive to the elevation $\alpha_1(k)$ and azimuth $\alpha_2(k)$ locations of the targets related to the inertial positions $\vec{p}(k)$ through regular coordinate transformations. Figure 9.1 shows the cross-array geometry of the sensor arrangement indexed by the sequence in which sensor outputs are placed in the data vector. The data collected at the 64-element sensor array at

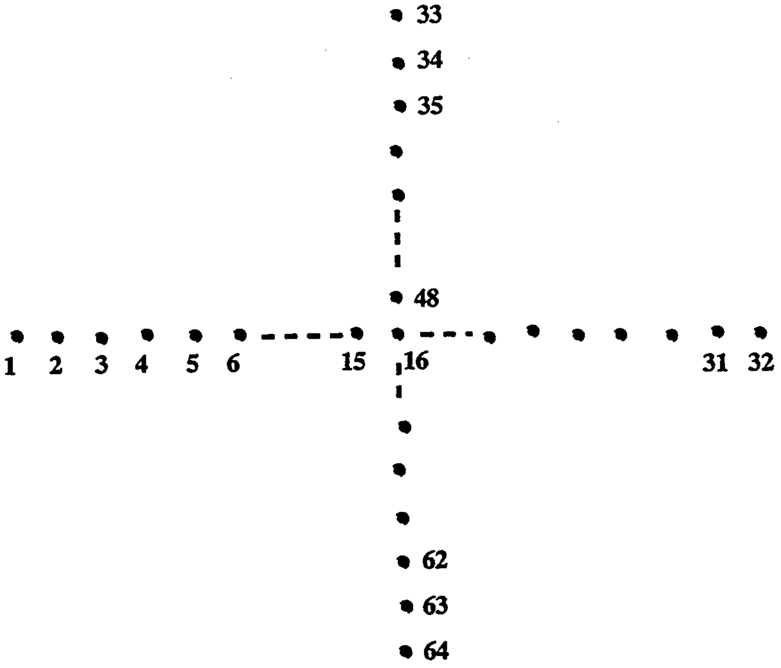


Figure 9.1: The figure shows the indexed sequence of sensor elements arranged in cross-array geometry.

time t is the superposition of incoming signal and ambient noise,

$$\vec{y}_1(k) = \vec{d}(\vec{p}(k))s(k) + \vec{n}_1(k), \tag{9.1}$$

where $\vec{n}_1(k)$ is a 64×1, 0-mean complex gaussian random vector of the Goodman class with covariance $\sigma_1^2 I$, $s(k)$ is the complex signal value and $\vec{d}(\vec{p}(k))$ is a regular 64×1 vandermonde direction vector with the angles of signal arrival parameterized by inertial postitons $\vec{p}(k)$.

The angular location of the target is reflected in the phase differences of signals reaching various sensors depending upon the array geometry encoded by the vandermonde direction vector. The phases at the sensor elements are measured with the 16th sensor (central) being the reference. The direction vector for the angular location α_1 (elevation), α_2 (azimuth) determines the array manifold and is given by a 64-element vector

$$\vec{d}(\alpha_1, \alpha_2) = \begin{bmatrix} e^{-15\pi cos(\alpha_1)cos(\alpha_2)} \\ \vdots \\ e^{-\pi cos(\alpha_1)cos(\alpha_2)} \\ 1 \\ \vdots \\ e^{16\pi cos(\alpha_1)cos(\alpha_2)} \\ e^{-16\pi cos(\alpha_1)sin(\alpha_2)} \\ \vdots \\ e^{-\pi cos(\alpha_1)sin(\alpha_2)} \\ e^{\pi cos(\alpha_1)sin(\alpha_2)} \\ \vdots \\ e^{16\pi cos(\alpha_1)sin(\alpha_2)} \end{bmatrix}$$

Using the tranformation between the polar and rectangular coordinates the direction vector can be written as function of the target postions in rectangular coordinates, i.e. $\vec{d}(\vec{p}(k)) = \vec{d}(\alpha_1(\vec{p}(k)), \alpha_2(\vec{p}(k)))$.

Appendix 9.2 Airplane Motion Based on Rigid Body Dynamics

Airplane motion is studied using the rigid body dynamics following the analysis in [22]. The translational position is represented in the inertial frame in both rectangular coordinates $\vec{p}(s) = [p_1(s)\ p_2(s)\ p_3(s)] \in R^3$ and polar coordinates,

$[r(s)\ \alpha_1(s)\ \alpha_2(s)]$, (range, elevation and azimuth) with the standard relationship

$$\begin{aligned} r &= \sqrt{p_1^2+p_2^2+p_3^2}\ , \\ \alpha_1 &= arctan\frac{p_3}{\sqrt{p_1^2+p_2^2}}\ , \\ \alpha_2 &= arctan\frac{p_2}{p_1}\ . \end{aligned} \tag{9.2}$$

Motion is described in terms of the components of its velocity vector projected along the body axes, $\vec{v}(s) = [v_1(s)\ v_2(s)\ v_3(s)] \in R^3$. The inertial positions $\vec{p}(s)$ are related to these velocities through the transformation

$$\vec{p}(s) = \vec{p}(t_0) + \int_{t_0}^{s} \Phi(\tau)\vec{v}(\tau)d\tau, \tag{9.3}$$

where $\vec{p}(t_0)$ is assumed known and Φ is a 3×3 time varying rotation matrix (given by Eqn 2.1) determined by rotational motion expressed via the Euler angles $\vec{\phi}(s) = [\phi_1(s)\ \phi_2(s)\ \phi_3(s)] \in \mathcal{M}(3)$, (pitch, roll and yaw).

Finally , the rotational motion determines the prior since the angular velocity projections, $\vec{q}(s) = [q_1(s)\ q_2(s)\ q_3(s)]$, onto the rotating body axes determine the rotation matrix $A_1(\vec{\phi}(s), \dot{\vec{\phi}}(s))$ (Eqn 3.1.1). The vector $\vec{q}(s)$ is dependent on the Euler angles and their derivatives according to,

$$\begin{aligned} q_1 &= \dot{\phi}_1 - \dot{\phi}_3 sin(\phi_2), \\ q_2 &= \dot{\phi}_2 cos(\phi_1) + \dot{\phi}_3 cos(\phi_2) sin(\phi_1), \\ q_3 &= -\dot{\phi}_2 sin(\phi_1) + \dot{\phi}_3 cos(\phi_2) cos(\phi_1) \end{aligned} \tag{9.4}$$

Covariance of Body-Frame Velocities $\vec{v}(s)$

Now we derive the covariance function of the airplane's inertial positions using the newtonian equations of motion. The solution of the vector equation 3.3 is given by

$$\vec{v}(s) = \int_{t_0}^{s} e^{-\int_{\tau_1}^{s} A(\tau)d\tau} \vec{f}(\tau_1)d\tau_1 + e^{-\int_{t_0}^{s} A(\tau)d\tau}\vec{v}(t_0)$$

Assuming known initial velocity $\vec{v}(t_0)$, the velocity covariance function can be found as

$$\begin{aligned}
\mathcal{K}_v(s_1, s_2) &= \int_{t_0}^{s_1}\int_{t_0}^{s_2} [e^{-\int_{\tau_1}^{s_1} A(\tau)d\tau}]\mathcal{E}\{\vec{f}(\tau_1)\vec{f}^{\dagger}(\tau_2)\}[e^{-\int_{\tau_2}^{s_2} A(\tau)d\tau}]^{\dagger} d\tau_1 d\tau_2 \\
&= \sigma_0^2 \int_{t_0}^{min(s_1,s_2)} [e^{-\int_{\tau_1}^{s_1} A(\tau)d\tau}][e^{-\int_{\tau_1}^{s_2} A(\tau)d\tau}]^{\dagger} d\tau_1
\end{aligned}$$

since the force vector $\vec{f}(s)$ is assumed 0-mean Gaussian with covariance $\sigma_0^2 I$.

Appendix 9.3 Jump and Diffusion Processes

9.3.1 Jump Processes

A detailed treatment on jump processes is presented in [36].

Definition: If $X(s)$ is a Markov process and satisfies

- $X(s)$ is continuous in probability, i.e.

$$\lim_{s \downarrow t} P(t, x, s, A) = \lim_{s \downarrow t} P_{s-t}(x, A) = 1_A(x)$$

- The function $q(x, A)$ (for $x \notin A$) defined by

$$\lim_{t \downarrow 0} \frac{1}{t} P(X_t \in A | X_0 = x) = q(x, A), \quad \text{converges uniformly in } \; t, x, A$$

for all $x \notin A$,

then $X(s)$ is a jump process with intensity $q(x, A)$.

The generator, or the backward Kolmogorov operator, for the jump process is given by

$$Af(x) = q(x) \int_E Q(x, dy)(f(y) - f(x)) \tag{9.5}$$

where $Q(x, dy)$ satisfies

- $\int_{E\ /\{x\}} Q(x, dy) = 1, \forall dy : x \notin dy$.
- $Q(x, dx) = 0$.
- $q(x, dy) = q(x)Q(x, dy)$

where $f(x)$ belongs to the domain of the operator A.

9.3.2 Diffusion Processes

There is some disagreement in the recent literature as to the definition of a diffusion process. The definition presented here is obtained from [37, 38].

Definition: A *Diffusion Process* is a sample path continuous Markov process with the transition funtion $P(s, x; t, B)$ which satisfies the following properties For every $\epsilon > 0$,

-

$$\lim_{t \downarrow 0} \frac{1}{t} \int_{|x-y|>\epsilon} P(s, x; t + s, dy) = 0,$$

- there exists a Borel measureable function $a(s,x)$ called the infinitesimal mean, such that

$$\lim_{t\downarrow} \frac{1}{t} \int_{|x-y|\leq\epsilon} (y-x)P(s,x;s+t,dy) = a(s,x),$$

- there exists a (positive semi-definite) matrix $B(s,x)$ called the infinitesimal variance or the diffusion matrix such that

$$\lim_{t\downarrow} \frac{1}{t} \int_{|x-y|\leq\epsilon} (y-x)^{\dagger}(y-x)P(s,x;s+t,dy) = B(s,x$$

For a homogeneous process the infinitesimal mean and variance are indepedent of time, i.e. $a(x,t) = a(x)$ and $B(x,t) = B(x)$. In the semi-group theory, the Markov processes are characterized by the infinitesimal generator of the semi-group associated with the process. This generator or the Backward Kolmogorov operator is defined as a linear operator A ,$A : \mathcal{D}(A) \subseteq \mathcal{B} \rightarrow \mathcal{B}$ such that

$$(Af)(x) = \lim_{h\downarrow 0}(\int P(s,x;s+h,dy)f(y) - f(x))$$

where the limit is define in the sup-norm ($||.||_\infty$ norm). The domain $\mathcal{D}(A)$ consists of all the functions $f \in \mathcal{B}$ for which this limit exists. For a diffusion the generator is given by

$$Af(x) = a(x) \circ \nabla f(x) + \frac{1}{2} \sum_{i,j=1}^{n} b(x)_{ij} \frac{\partial^2 f(x)}{\partial x_i \partial x_j}$$

where $\circ$ stands for the vector dot-product.

Appendix 9.4 Proof of Theorem 1

Proof: This analysis is carried out for a fixed t so we drop the subscript t without any ambiguity. There are essentially two points to the proof: (1) showing that $\pi(\vec{x}(M))$ is an invariant density of the process, and (2) verifying that the process is irreducible and therefore $\pi(\vec{x}(M))$ is the unique invariant density. Part (2) follows directly that in [13, 12] using the properties of the jump process and from the fact that the diffusions are each irreducible over their respective subspaces. In part (1) we need to verify the stationarity for both the jump and diffusion components of the Markov process. The generator, or backward Kolmogoroff operator, for the jump-diffusion process (denote it as $A = A^d + A^j$ (diffusion+jump)) characterizes the stationary density in that $\pi(\vec{x}(M))$ is stationary for the jump-diffusion if and only if $\int Af(\vec{x}(M))\pi(\vec{x}(M))d\vec{x}(M) = 0$ for all f in the domain of A, $\mathcal{D}(A)$.

The diffusion process has two components corresponding to the S.D.E on the multi-dimensional torus (4.4) and the S.D.E. on the Euclidean space of target positions (4.5). To prove invariance of $\pi(\vec{x}(M))$ for the diffusion on torus we use results from [12] on invariant distributions of S.D.E.'s on linear manifolds, in particular the multi-dimensional Torus. The stationarity condition is verified for the Euclidean component as follows. Define a set of functions which forms the domain of the generator A as

$$\mathcal{D}(A) = \{f : f = \sum_{m=0}^{M} 1_{\mathcal{X}(m)} f_m, f_m \in \hat{C}^2(\mathcal{X}(m)), M \geq 0\} . \tag{9.6}$$

Then the infinitesimal generator for diffusion A^d becomes

$$A^d f(\vec{x}(M)) = -\frac{1}{2}\left(\nabla_2 E(\vec{x}(M)) \circ \nabla_2 f(\vec{x}(M))\right) + \frac{1}{2}\sum_{i=1}^{3n_M} \frac{\partial^2 f(\vec{x}(M))}{\partial^2(\vec{p}(M))_i} \;, \forall f \in \mathcal{D}(A)$$

where $n_M = \sum_{m=1}^{M}$, total number of track-segments in the parameter set $\vec{x}(M)$, $\circ$ stands for the vector dot-product and the gradients $\nabla_2 E(\vec{x}(M)), \nabla_2 f(\vec{x}(M))$ are w.r.t the position vector $\vec{p}(M)$. Substituting this expression in the integral condition we get

$$\begin{aligned}\int A^d f(\vec{x}(M))\pi(\vec{x}(M))d\vec{x}(M) \;&=\\ &-\int \frac{1}{2}(\nabla E(\vec{x}(M)) \circ \nabla f(\vec{x}(M))\frac{e^{-E(\vec{x}(M))}}{\mathcal{Z}}d\vec{x}(M)\\ &+\int \frac{1}{2}\left(\sum_{i=1}^{3n_M} \frac{\partial^2 f(\vec{x}(M))}{\partial^2(\vec{p}(M))_i}\right)\frac{e^{-E(\vec{x}(M))}}{\mathcal{Z}}d\vec{x}(M)\;.\end{aligned}$$

Integration by parts of the second term, with the fact that the function f vanishes at the boundary, results in a term which is negative of the first term. Therefore the given posterior $\pi(\vec{x}(M))$ is the stationary density of the diffusion process.

The generator A^j of a jump process is given by (Eqn. 9.5)

$$A^j f(\vec{x}(M)) = q(\vec{x}(M)) \int_{\mathcal{X}(\mathcal{T}^1(\vec{x}(M)))} Q(\vec{x}(M), dy)(f(y) - f(\vec{x}(M)))\;.$$

When substituted in the stationarity condition, it provides

$$q(\vec{x}(M))\pi(\vec{x}(M))d\vec{x}(M) = \int_{\mathcal{X}(\mathcal{T}^{-1}(\vec{x}(M)))} q(\vec{y}, d\vec{x}(M))\pi(\vec{y})d\vec{y}\;,$$

which is often called as the *detailed balance condition.* Therefore, the jump parameters should satisfy this equation for the density $\pi(\vec{x}(M))$ to be the stationary density of the jump process. We will prove this condition assuming only birth/death of tracks, the treatment for birth/deaths of track segments being similar. Substituting for the transition measures in the *detailed balance condition* and simplifying we obtain,

$$\begin{aligned}
& \pi(\vec{x}(M))d\vec{x}(M) \; [\; \sum_{j=1}^{M+1} \int_{\mathcal{X}(1)} q_T^b(\vec{x}(M), \vec{x}(M) \oplus_j \vec{y}(1)) d\vec{y}(1) + \sum_{j=1}^{M} q_T^d(\vec{x}(M), \wp_T^{(j)} \vec{x}(M)) \;] \\
= \quad & d\vec{x}(M) \; [\; \sum_{j=1}^{M+1} \int_{\mathcal{X}(1)} q_T^d(\vec{x}(M) \oplus_j \vec{y}(1), \vec{x}(M)) \pi(\vec{x}(M) \oplus_j \vec{y}(1)) d\vec{y}(1) \\
& + \sum_{j=1}^{M} q_T^b(\wp_T^{(j)} \vec{x}(M), \vec{x}(M)) \pi(\wp_T^{(j)} \vec{x}(M)) \;]
\end{aligned} \tag{9.7}$$

Substituting the values for q_T^b, q_T^d from 4.3, and analyzing only the j^{th} terms from sums on both sides,

R.H.S.

$$\begin{aligned}
& \frac{d\vec{x}(M)}{Z_T(1)} \; [\int_{\mathcal{X}_1} \frac{1}{4(M+1)} e^{-[L(\vec{x}(M)) - L(\vec{x}(M) \oplus_j \vec{y}(1))]+} \pi(\vec{x}(M) \oplus_j \vec{y}(1)) d\vec{y}(1) \\
& \quad + \frac{1}{4M} e^{-[L(\vec{x}(M)) - L(\wp_T^{(j)} \vec{x}(M))]+} e^{-P(\vec{x}^{(j)}(1))} \pi(\wp_T^{(j)} \vec{x}(M)) \;] \\
= \quad & \frac{d\vec{x}(M)}{(\mathcal{Z})(Z_T(1))} \; [\; \frac{1}{4(M+1)} \int_{\Omega_>} e^{-L(\vec{x}(M))} e^{-P(\vec{x}(M) \oplus_j \vec{y}(1))} d\vec{y}(1) \\
& \quad + \frac{1}{4(M+1)} \int_{\Omega_\leq} e^{-L(\vec{x}(M) \oplus_j \vec{y}(1))} \; e^{-P(\vec{x}(M) \oplus_j \vec{y}(1))} d\vec{y}(1) \\
& \quad + \frac{1}{4M} e^{-[L(\vec{x}(M)) - L(\wp_T^{(j)} \vec{x}(M))]+} e^{-P(\vec{x}^{(j)}(1))} e^{-L(\wp_T^{(j)} \vec{x}(M))} e^{-P(\wp_T^{(j)} \vec{x}(M))} \;] \; ,
\end{aligned} \tag{9.8}$$

where

$$\Omega_> = \mathcal{X}(1) \bigcap \{ \vec{y}(1) : L(\vec{x}(M)) > L(\vec{x}(M) \oplus_j \vec{y}(1)) \}$$

Substituting this expression in the stationarity condition,

$$\begin{aligned}\int Af(\vec{x}(M))\pi(d\vec{x}(M))d\vec{x}(M) &= \int\{-[K_p\nabla_2 L(\vec{x}(M))+\vec{p}(M)]\circ\nabla_2 f(\vec{x}(M)) \\ &\sum_{i,j=1}^{3n_M}[K_p\hat{\circ}\nabla_2^2 f(\vec{x}(M))]_{ij}\}\frac{1}{\mathcal{Z}}e^{-(L(\vec{x}(M))+\frac{1}{2}\vec{p}(M)^\dagger K_p^{-1}\vec{p}(M))}d\vec{x}(M)\ .\end{aligned}$$

Integrating by parts the second term on right side and using boundary conditions we obtain,

$$\frac{1}{\mathcal{Z}}\int\left[\nabla_2 f(\vec{x}(M))^\dagger\circ\{K_p\nabla_2 L(\vec{x}(M))+\vec{p}(M)\}\right]e^{-(L(\vec{x}(M))+\frac{1}{2}\vec{p}(M)^\dagger K_p^{-1}\vec{p}(M))}d\vec{x}(M)\ ,$$

which is negative of the first term in the equation.

Q.E.D

Appendix 9.6 Proof of Theorem 2

The jump-diffusion Markov process is constructed on a multi-dimensional torus similar to the $X_2(s)$ component of the process $X(s)$ in theorem 1. With only the change in dimensions of the torus the results for the diffusion component are again referred to [12] as in proof of theorem 1. In this case, the jump moves are performed based on the Gibb's sampling instead of the Metropolis algorithm as before, therefore, we need to verify the stationarity condition for the jump process. Again, we will prove this condition assuming only birth/death of tracks, the treatment for birth/deaths of track segments being of similar nature. Using the intensities stated in the theorem, the detailed balance condition is verified as follows (similar to Eqn 9.7).

$$\begin{aligned} & \pi(\vec{c}(M))d\vec{c}(M)\ [\ \sum_{j=1}^{M+1} \int_{C(1)} q_T^b(\vec{c}(M), \vec{c}(M) \oplus_j \vec{c}(1))d\vec{c}(1) + \sum_{j=1}^{M} q_T^d(\vec{c}(M), \wp_T^{(j)}\vec{c}(M))\ ] \\ = \quad & d\vec{c}(M)\ [\ \sum_{j=1}^{M+1} \int_{C(1)} q_T^d(\vec{c}(M) \oplus_j \vec{c}(1), \vec{c}(M))\pi(\vec{c}(M) \oplus_j \vec{c}(1))d\vec{c}(1) \\ & + \sum_{j=1}^{M} q_T^b(\wp_T^{(j)}\vec{c}(M), \vec{c}(M))\pi(\wp_T^{(j)}\vec{c}(M))\ ] \end{aligned}$$

After substituting the intensities q_T^b, q_T^d from the theorem statement and analyzing only the j^{th} terms from the sums, both the sides reduce to

$$\pi(\vec{c}(M))d\vec{c}(M)[\ \int_{C(1)} \pi(\vec{c}(M) \oplus_j \vec{c}(1))d\vec{c}(1) + \pi(\wp_T^{(j)}\vec{c}(M))\ ]\ .$$

Q.E.D

10. GLOSSARY

$\vec{v}(k)$	the velocity vector projected onto the body frame of reference, $\in \Re^3$.
$\vec{p}(k)$	the vector of target position in inertial frame at time k, $\in \Re^3$.
$\vec{q}(k)$	the vector of angular velocities of the airplane projected on to the body axes.
$\mathcal{M}(3)$	the parameter space of the airplane orientations given by the torus $[0, 2\pi]^3$ with 0 identified to 2π.
$\mathcal{A}$	the discrete alphabet set containing symbols for all target types.
$\vec{\phi}(k)$	the vector of Euler's angles (pitch, roll and yaw) representing target's orientation with respect to the inertial frame, $\in \mathcal{M}(3)$.
$x^{(m)}(k)$	the parameter set associated with target m at time $k < t$, $\{a(k), \vec{\phi}(k), \vec{p}(k)\} \in (\mathcal{M}(3) \times \mathcal{A} \times \Re^3)$.
$t_1^{(m)}, t_1^{(m)} + t^{(m)}$	the times of appearance and disappearnce of the m^{th} target, its dwell time is given by $\mathcal{T}^{(m)} = [t_1^{(m)} + 1, .., t_1^{(m)} + t^{(m)}]$.
$\vec{x}_t(M)$	the set of parameters defining a scene containing M-tracks for the observations up to time t, $\vec{x}_t(M) = \bigcup_{m=1}^{M} \{\vec{x}^{(m)}(k) : k \in \mathcal{T}^{(m)}\} \quad \in \mathcal{X}_t(M)$.
$\mathcal{X}_t(M)$	the parameter subspace associated with M targets, $\equiv \prod_{m=1}^{M} \left(\bigcup_{t^{(m)}=1}^{t} (\mathcal{M}(3) \times \mathcal{A} \times \Re^3)^{t^{(m)}} \times \aleph \right)$.
$\mathcal{X}_t$	the complete parameter space for obersvations up to time t, $\bigcup_{M=0}^{\infty} \mathcal{X}_t(M)$.
$\vec{f}(t)$	the vector of linear forces driving the airplane along its body axes.
$P_t(\vec{x}_t(M))$	the Gibb's energy corresponding to the prior density.

$L_t(\vec{x}_t(M))$	the Gibb's energy corresponding to the likelihood of the data collected up to time t.
$E_t(\vec{x}_t(M))$	the Gibb's energy corresponding to the posterior density conditioned on the data collected up to time t, $E_t(\vec{x}_t(M)) = P_t(\vec{x}_t(M)) + L_t(\vec{x}_t(M))$.
$\vec{d}(\vec{p}^{(m)}(k))$	the 64-element vandermonde direction vector for the m^{th} target at the location $\vec{p}^{(m)}(k)$.
$D(\vec{p}^{(1)}(k),..\vec{p}^{(M)}(k))$	the $64 \times M$ direction matrix having a column of vandermonde directior vector for each target.
$\vec{y}_1(k)$	the 64 length complex valued data vector recorded by tracking array at time k.
$\vec{y}_2(k)$	the set of 64×64 grey scale pixel values generated by imaging radar at time k .
$I_t^{\mathcal{P}}(1)$	the data collected by tracking array up to time t, $I_t^{\mathcal{P}}(1) \equiv \{\vec{y}_1(k) : k \in \{1,.,t\}\}$.
$I_t^{\mathcal{P}}(2)$	the imaging data collected up to time t, $I_t^{\mathcal{P}}(t) \equiv \{\vec{y}_2(k) : k \in \{1,..,t\}\ \}$.
$I_t^{\mathcal{P}}$	the complete data set up to time t, $I_t^{\mathcal{P}} = \{I_t^{\mathcal{P}}(1), I_t^{\mathcal{P}}(1)\}$.
$\mathcal{P}(\cdot)$	the far field orthographic projection transformation used to model the high resolution optical imaging sensor.
$\vec{n}_1(k)$	the P-element vector of i.i.d random variables each having the density $\sim N(0, \sigma_1^2)$ representing noise at the tracking array at time k.
Inertial Frame	the reference frame fixed to ground based sensor systems.
Body-Fixed Frame	the reference frame centered at the origin of inertial frame but parallel to airplane's body axes.

11. Bibliography

[1] Y. Bar-Shalom and E. Tse. Tracking in cluttered environment with probabilistic data association. *Automatica*, (11):451–460, 1975.

[2] Y. Bar-Shalom and T. E. Fortmann. *Tracking and Data Association.* Academic Press, 1988.

[3] Editor: Y. Bar-Shalom. *Multitarget-Multisensor Tracking.* Artech House, 1990.

[4] Chang and Chaw-Bing. Target tracking using state estimation; survey of problems and solutions. *IEEE Transactions on Automatic Control*, pages 98–109, February 1984.

[5] C. R. Rao, C. R. Sastry, and B. Zhou. Tracking the direction of arrival of multiple moving targets. *IEEE Transactions on Acoustics,Speech and Signal Processing*, accepted for publication.

[6] C. R. Sastry, E. W. Kamen, and M. Simaan. An efficient algorithm for tracking angles of arrival of moving targets. *IEEE Transactions on Acoustics,Speech and Signal Processing*, ASSP-39(No.1):242–246, 1991.

[7] C. K. Sword, M. Simaan, and E. W. Kamen. Multiple target angle tracking using using sensor array outputs. *IEEE Trans. AES.*, 26(2):367–373, 1990.

[8] A. Satish and R. L. Kashyap. Maximum likelihood based algorithm for multiple target tracking. In *Proc. 31th Annual Allerton Conference on Communication, Control, and Computing*, Urbana, Il, September 1993. University of Illinois.

[9] A. L. Swindlehurst and T. Kailath. Passive direction-of-arrival and range estimation for near-field sources. In *Proc. IEEE ICASSP*, pages 123–128, 1988.

[10] D. D. Sworder, P. F. Singer, D. Doria, and R. G. Hutchins. Image-enhanced estimation methods. *Proc. of the IEEE*, 81:796–812, June 1993.

[11] U. Grenander and M. I. Miller. Jump-diffusion processes for abduction and recognition of biological shapes. *Monograph of the Electronic Signals and Systems Research Laboratory*, 1991.

[12] Y. Amit, U. Grenander, and M.I. Miller. Ergodic properties of jump-diffusion processes. *Annals of Applied Probability*, submitted December 1992.

[13] U. Grenander and M. I. Miller. Representations of knowledge in complex systems. *Journal of the Royal Statistical Society*, to appear October, 1993.

[14] R. Schmidt. *A signal subspace approach to multiple emitter location and spectral estimation.* Ph.D. Dissertation of Stanford University, Palo Alto, CA., Nov. 1981.

[15] M.I. Miller and D. R. Fuhrmann. Maximum likelihood narrow-band direction finding and the em algorithm. *IEEE Acoust. Speech and Signal Processing*, 38, No.9(38, No.9):560–577, 1990.

[16] A. Srivastava, M. I. Miller, and U. Grenander. Jump-diffusion processes for object tracking and direction finding. In *Proc. 29th Annual Allerton Conference on Communication ,Control and Computing*, pages 563–570, Urbana, Il, October 1991. University of Illinois.

[17] A. Srivastava, N. Cutaia, M. I. Miller, J. A. O'Sullivan, and D. L. Snyder. Multi-target narrowband direction finding and tracking using motion dynamics. In *Proc. 30th Annual Allerton Conference on Communication, Control, and Computing*, pages 279–288, Urbana, Il, October 1992. University of Illinois.

[18] M.I. Miller, R. S. Teichman, A. Srivastava, J.A. O'Sullivan, and D. L. Snyder. Jump-diffusion processes for automated tracking-target recognition. In *1993 Conference on Information Sciences and Systems*, Baltimore, Maryland, March 24-26 1993. Johns Hopkins University.

[19] A. Srivastava, R. S. Teichman, and M. I. Miller. Target tracking and recognition using jump-diffusion processes. In *Proc. ARO's Eleventh Army Conference on Applied Mathematics and Computing*, Carnegie Melon University, Pittsburgh, PA, June 1993.

[20] W. Freiberger and U. Grenander. Computer generated image algebras. *Internation Federation Information Processing*, 68:1397–1404, 1969.

[21] U. Grenander. Advances in pattern theory: The 1985 rietz lecture. *The Annals of Statistics*, 17:1–30, 1985.

[22] Bernard Friedland. *Control System Design : An Introduction To State-Space Methods.* McGraw-Hill Book Company, 1986.

[23] Y. Amit, U. Grenander, and M. Piccioni. Structural image restoration through deformable templates. *J. American Statistical Association*, 1991.

[37] I. Karatzas and S.E. Shreve. *Brownian Motion and Stochastic Calculus.* Springer-Verlag, 1987.

[38] E. Wong and B. Hajek. *Stochastic Processes in Engineering Systems.* Springer-Verlag, New York Berlin Heidelberg Tokyo, 1985.

[39] H. Kunita. Stochastic differential equations and stochastif flows of diffeomorphisms. In *École d' Été de Probabilités de Saint-Flour, XII -1982*, number 1097. Springer-Verlag L.N.M., 1984.

12. Vita

Anuj Srivastava

Date of Birth	9 September 1968
Place of Birth	Faridabad, India
Undergraduate Study	Institute of Technology Banaras Hindu University B.Tech.(honours) Electronics Engineering , May 1990
Graduate Study	Washington University, St. Louis, Missouri M.S. Electrical Engineering, October 1993
Honours and Awards	EE Doctoral Fellowship
Publications	Srivastava, Miller, Grenander. *Jump-Diffusion Processes for Object Tracking and Direction Finding.* Proc. 29th Annual Allerton Conference on Communication,Control and Computing.University of Illinois, Urbana, Il. Srivastava, Teichman, Miller. *Target Tracking and Recognition Using Jump-Diffusion processes.* Proc. ARO's Eleventh Army Conference on Applied Mathematics and Computing. Carnegie Melon University, Pittsburgh, PA.

December, 1993

Conditional-Mean Estimation Via Jump-Diffusion Processes in Multiple Target Tracking/Recognition [*]

M. I. Miller[†] and A. Srivastava[†] and U. Grenander[‡]

Abstract

A new algorithm is presented for generating the conditional mean estimates of functions of target positions, orientation and type in recognition and tracking of an unknown number of targets and target types. Taking a Bayesian approach a posterior measure is defined on the tracking/target parameter space by combining the narrowband sensor array manifold model with a high resolution imaging model, and a prior based on airplane dynamics. The Newtonian force equations governing rigid body dynamics are utilized to form the prior density on airplane motion. The conditional mean estimates are generated using a random sampling algorithm based on *Jump-Diffusion* processes, [1], for empirically generating MMSE estimates of functions of these random target positions, orientations and type under the posterior measure. Results are presented on target tracking and identification from an implementation of the algorithm on a networked Silicon Graphics and DECmpp/MasPar parallel machines.

---

[*] Submitted to the *IEEE Transactions on Signal Processing.* This paper was presented in part at the 29th Annual Allerton Conference on Communication, Control and Computing, 1991, University of Illinois at Urbana-Champaign, Illinois, and at the Communications and Informations Sciences and Systems Conference at Johns Hopkins University, March 1993. This work was supported by the ARO DAAL03-92-G-0141, ONR N00014-92-J-1418, ONR N00014-94-1-0859, and Rome Laboratory F30602-92-C-0004 to Michael I. Miller and ARO/MIT DAAL03-92-G-0115, ONR N00014-91-J-1021, ARL MDA972-93-1-0012, and NSF DMS-9217655 to Ulf Grenander.

[†] Department of Electrical Engineering, Electronic Signals and Systems Research Laboratory, Washington University St. Louis, MO. 63130

[‡] Division of Applied Mathematics, Brown University, Providence, Rhode Island

1 Introduction

This paper focuses on automated tracking and recognition of objects in remotely sensed complex dynamically changing scenes. Grenander's global shape models are used herein, extended to parametric representations of arbitrary and unknown model order, in which typical shape is represented via templates, with variability represented via transformation groups applied to the templates. The types of variability associated with the classical geometry are accommodated via the *Euclidean groups* involving both the rigid motions of translation and rotation. Since the objects are under dynamic motion, *the parameter spaces involves Cartesian products of these similarity groups.*

The second fundamental type of variability is associated with the model order (parametric dimension) and model type (recognition). In any scene there may be variable numbers of and different kinds of targets existing in the scenes for varying periods of time, implying the target number and therefore parametric dimension are unknown apriori. *Hence, the inference or hypothesis space becomes a search across countable disconnected unions of these Cartesian product groups*, with the model order and model type a variable to be inferred. We take a Bayesian approach, i.e. we define a prior distribution supported on this countable union of spaces, from which the posterior distribution is constructed. The parametric representation of the target scene is selected to correspond to *conditional expectations* under this posterior.

As we are particularly interested in non-cooperative moving targets, the algorithms are made robust to motion by incorporation of knowledge about motion dynamics into the prior distribution. The Newtonian force equations, a system of differential equations governing the motion of targets are used to induce the prior. These differential equations are parameterized by the target and or sensor type, and its orientation motion described by rotations in the 3-dimensional torus group. It is the introduction of these Newtonian force equations which makes tracking and recognition inseparable, since the equations of motion are explicitly parameterized by the sequence of airplane orientations. This provides the significant link between tracking algorithms based on data from narrowband sensors arrays in which the target is unresolved in the data (effectively a point), and high resolution information perhaps provided by a second sensor preserving the orientation information from which target recognition is performed. In part, it is this fundamental link which has motivated us to solve the tracking/recognition problem in a single consistent estimation framework in which the inference proceeds via the fusion of multi-sensor data: in our case, a *narrowband sensor array output* and *high-resolution images.*

Concerning the generation of conditional expectations, except under the most simplifying set of assumptions, the posterior distribution will be highly nonlinear in the parameters of hypothesis space, thus, precluding the direct closed form analytic generation of conditional expectations. Towards this end we have taken advantage of the explosion which has occurred over the past 10 years in the statistics community on the introduction of random sampling methods for the empirical generation of estimates from complicated distributions; see for example the reviews [2, 3]. Motivated by such approaches, we have previously described a new family of random sampling algorithms [4, 1] for generating conditional expectations in such disconnected hypothesis spaces. The random samples are generated via the direct simulation of a Markov process whose state moves through the hypothesis space with the *ergodic property* that the transition distribution of the Markov process converges to the posterior distribution. This allows for the empirical generation of conditional expectations under the posterior. To accommodate the connected and disconnected nature of the state spaces, the Markov process is forced to satisfy *jump-diffusion dynamics*. i.e. through the connected parts of the parameter space (Lie manifolds) the algorithm searches continuously, with sample paths corresponding to solutions of standard diffusion equations; across the disconnected parts of parameter space the jump process determines the dynamics. The infinitesimal properties of these jump-diffusion processes are selected so that various sample statistics converge to their expectation under the posterior.

The original motivation for introducing jump-diffusions in [4, 1] is to accommodate the very different continuous and discrete components of the object discovery process. Given a conformation associated with a target type, or group of targets, the problem is to identify the orientation and translation parameters accommodating the variability manifest in the viewing of each object type. For this, the parameter space is sampled using diffusion search in which the state vector winds continuously through the similarities following gradients of the posterior. The second distinct part of the sampling process corresponds to the target type and number deduction during which the target types are being discovered, with some subset of the scene only partially "recognized" at any particular time during the process. The second type of change in parameter space are associated with a set of non-continuous transformations of the scene controlled by the jump process. A jump in hypothesis space corresponds to (i) jumping between different object types, (ii) hypothesizing a new object in the scene or a "change of mind" via the deletion of an object in the scene, or (iii) the merging or splitting of tracks and objects. The jump intensities are governed by the posterior density, with the process visiting configurations of higher probability for longer exponential times, and the diffusion equation governing the dynamics between jumps. It is the fundamental difference between diffusions (almost surely continuous sample paths) and jump processes (making large moves in parameter space in small time) which allows us to explore the very different connected and non-connected nature of hypothesis space.

Now automated target tracking and recognition are well known problems in the signal processing and control system's literature, with a great deal of published work on multiple target tracking posed as state estimation problems [5, 6, 7]. In such approaches Kalman filter based techniques are emphasized, with linear descriptions of state playing a fundamental role. For situations in which the observed data are non-linear in target parameters the use of the extended Kalman filter has been proposed corresponding to linear approximations which prove valid for particular scenarios. There also now exists a substantial body of important work in tracking the directions of arriving signals from multiple moving sources recorded via sensor arrays [8, 9, 10]. In such sensor array based approaches the non-linear relationship between the parameters of motion and the sensor data are addressed directly, the linear Kalman filter state equations for tracking guiding or providing initial conditions for the gradient based estimators generated from the likelihood. In these non-linear data models, several variations of the gradient based techniques are used to solve the problem in mostly maximum-likelihood settings. However, the majority of researchers utilize simplifying assumptions which are not always valid in a general tracking scenario. For example, targets may be assumed stationary between sample times with multiple (~ 100) snapshots at each sample time, whereas, in general, for a moving target, each data sample reflects a new position. Also, though researchers base their models on simplified versions of target dynamics for the tracking scenario, mostly constant velocity - constant acceleration state constraint equations have been used because of their linear nature. These restricted motions are partly due to assumptions required for Kalman updating, but perhaps more fundamentally due to the separation of the tracking and recognition problems. The more informative priors used in this paper require high resolution recognition as the priors are coupled to the target type and its orientations. In part, this is one of the major results of this work.

In the work presented here, we define a random sampling based solution for generating minimum mean squared error estimates of the state variables for tracking and recognition problems in a general setting. We assume data from a narrowband sensor array providing azimuth-elevation data for object tracking, and optical or radar imagers providing detailed information about the target-type and orientation. The goal is to track and recognize the unknown number of non-cooperative sources. The paper is organized as follows. In section 2 we define the parameter spaces with the posterior distribution derived in section 3. Section 4 describes an inference algorithm based on jump-diffusion processes and section 5 presents various results.

2 Recognition Via Deformable Templates

We use the global shape models and pattern theoretic approach introduced by Grenander [11, 12] to analyze complex scenes. As the basic building blocks of the hypotheses we define a subset of generators $\mathcal{G}^o$, which contains each target type $a \in \mathcal{A}$ ($\mathcal{A}$ is the alphabet of target types) placed at the origin of the inertial reference frame aligned to the inertial axes. The fundamental variability in target space is accommodated by applying the transformations $T(\phi)$ and $T(p)$ to the templates $g^o \in \mathcal{G}^o$ according to

$$T(\phi) : \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi_1 & sin\phi_1 \\ 0 & -sin\phi_1 & cos\phi_1 \end{bmatrix} \begin{bmatrix} cos\phi_2 & 0 & -sin\phi_2 \\ 0 & 1 & 0 \\ sin\phi_2 & 0 & cos\phi_2 \end{bmatrix} \times \begin{bmatrix} cos\phi_3 & sin\phi_3 & 0 \\ -sin\phi_3 & cos\phi_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad (1)$$

$$T(p) : \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \rightarrow \begin{bmatrix} x_1 + p_1 \\ x_2 + p_2 \\ x_3 + p_3 \end{bmatrix}, \quad (2)$$

where $\phi \in [0, 2\pi]^3$ with $0, 2\pi$ identified (herein referred to as the 3-dimensional torus $\mathcal{T}(3)$), and $p \in R^3$ is the translation vector. These parameterized transformations operate on the templates from $\mathcal{G}^o$ generating the full set of possible elements constituting any scene. The left panel of Figure 1 shows a rendering of one of the 3-D ideal targets $g^o \in \mathcal{G}^o$ under one such transformation.

The Bayes posterior is parameterized via the set of transformations, as well as the airplane type. A pattern consisting of a single track arises from a single target appearing and disappearing at random times $t_1^{(m)}, t_2^{(m)} \in [t_0, t]$ the observation period, with the m-th track parameter vector $x^{(m)}$ an element of the space $x^{(m)} \in (\mathcal{X}_0 \bigcup \sharp)^{[t_0,t]} \times \mathcal{A}$, $\mathcal{X}_0 \equiv \mathcal{T}(3) \times \Re^3$. The symbol $\sharp$ is used to denote the absence of the target from the scene. It will be useful for us to introduce the notation $x^{(m)}(\tau), \tau \in [t_0, t]$ to denote the set of parameters encoding the m-th target at time τ. An M-track parameter vector $x(M)$ becomes

$$x(M) \in \mathcal{X}_t(M) \equiv \left[\left(\mathcal{X}_0 \bigcup \sharp \right)^{[t_0,t]} \times \mathcal{A} \right]^M \tag{3}$$

Since the number of the targets M is unknown a priori, the complete parameter space is defined as $\mathcal{X}_t = \bigcup_{M=0}^{\infty} \mathcal{X}_t(M)$. The estimation problem is to estimate the individual configurations as well as the number M.

3 Bayesian Posterior

Minimum mean squared error (*MMSE*) parameter estimates are generated via their empirical computation under the posterior measure. As the posterior is proportional to the product of the prior density and the observed data likelihood we first derive a prior on the parameter space followed by a model for the data generation which determines the posterior. For real time estimation problems, the posterior density is an explicit function of t denoted $\pi_t(\cdot)$. In Bayesian approach the estimates are for each time t conditioned on the data observed up to that time t.

3.1 Prior Density on Parameter Space $\mathcal{X}_t$

Airplane dynamics: The formulation of the prior measure on airplane positions is based on equations of motion for rigid bodies. We use an approach, in which the prior is induced via partial differential equations by assuming the forcing function to be a white process, which induces a Gaussian process with covariance corresponding to the differential operator expressing airplane dynamics. For this purpose, we use a formulation of airplane dynamics through differential equations as described by Cutaia and O'Sullivan [13]. Airplane dynamics are most straightforwardly expressed using the velocities projected along the body-fixed axes, called the body-frame velocities and here denoted $v(s) = [v_1(s)\ v_2(s)\ v_3(s)]$. They are depicted in the right panel of Figure 1.

Following standard rigid body analysis (see [14], for example) and neglecting the earth's curvature, motion and wind effects, the translational velocities $v(s)$ and rotational velocities $q(s) = [q_1(s)\ q_2(s)\ q_3(s)]$ satisfy the following set of differential equations:

$$
\begin{aligned}
\dot{v}_1(s) - q_3(s)v_2(s) + q_2(s)v_3(s) &= f_1(s)\,, \\
\dot{v}_2(s) + q_3(s)v_1(s) - q_1(s)v_3(s) &= f_2(s)\,, \\
\dot{v}_3(s) - q_2(s)v_1(s) + q_1(s)v_2(s) &= f_3(s)\,, \\
I_1\dot{q}_1(s) - (I_2 - I_3)q_2(s)q_3(s) &= \Gamma_1(s)\,, \\
I_2\dot{q}_2(s) - (I_3 - I_1)q_1(s)q_3(s) &= \Gamma_2(s)\,, \\
I_3\dot{q}_3(s) - (I_1 - I_2)q_2(s)q_1(s) &= \Gamma_3(s)\,,
\end{aligned}
\tag{4}
$$

where $[f_1(s)\ f_2(s)\ f_3(s)]$ is the vector of applied translational forces, $[I_1\ I_2\ I_3]$ is the vector of rotational inertias, and $[\Gamma_1(s)\ \Gamma_2(s)\ \Gamma_3(s)]$ is the vector of applied torques. The first three equations describe the airplane's translational motion, while the next three describe its rotational motion.

At this time the prior which we have used for tracking is "somewhat less informative" in that only the first three equations, on translational motion, are used; detailed models of the targets associated with the torques for describing the rotational motion are not yet explicitly incorporated. The system matrix $A(\phi(s), \dot{\phi}(s))$ parameterizing Eqns. 4 is

$$\begin{bmatrix} 0 & -q_3(s) & q_2(s) \\ q_3(s) & 0 & -q_1(s) \\ -q_2(s) & q_1(s) & 0 \end{bmatrix},$$

with the velocities, inertial positions and Euler angles are related using the standard transformation to relate body-frame velocities with inertial frame positions according to

$$p(s) = \int_{t_0}^{s} \Psi(\tau)v(\tau)d\tau + p(t_0), \tag{5}$$

where $p(t_0)$ the initial position is assumed known and $\Psi(\tau)$ is the standard orthogonal rotation matrix given in Eqn. 1 parameterized by the Euler angles $\phi(\tau)$. The rotational motion determines the prior since with reference to the fixed inertial frame the angular velocity projections, $q(s)$, onto the rotating body axes determine the system matrix $A(\phi(s), \dot{\phi}(s))$. The $q(s)$ vectors are none other than the rates of change of the Euler orientation angles according to $q_1 = \dot{\phi}_1 - \dot{\phi}_3 sin(\phi_2)$, $q_2 = \dot{\phi}_2 cos(\phi_1) + \dot{\phi}_3 cos(\phi_2) sin(\phi_1)$, $q_3 = -\dot{\phi}_2 sin(\phi_1) + \dot{\phi}_3 cos(\phi_2) cos(\phi_1)$.

For the construction of the "informative" part of the prior, first condition the linear differential equations on the sequence of system matrices $A(\phi(s), \dot{\phi}(s))$, via conditioning on the sequence of Euler rotation angles. Then the velocity process is a conditional Gaussian process induced by assuming the forcing function on the momentum equation to be a white process of fixed spectral density. The covariance function is derived as follows. Define the state transition matrix $\Phi(\tau, \cdot)$ as the unique solution of the matrix differential equation

$$\frac{dM(s)}{ds} = -A(\phi(s), \dot{\phi}(s))M(s) \ , \quad M(\tau) = I \ , \tag{6}$$

then the covariance of the body frame velocity process becomes

$$\mathcal{K}_v(s_1, s_2) = \sigma \int_{t_0}^{min(s_1,s_2)} \Phi(t_1, s_1)\Phi^{\dagger}(t_1, s_2)dt_1 + \Phi(t_0, s_1)\mathcal{K}_v(t_0, t_0)\Phi^{\dagger}(t_0, s_2) \ , \tag{7}$$

where $\mathcal{K}_v(t_0, t_0)$ is the covariance of the initial velocity, $v(t_0)$. The inertial position process is then Gaussian with covariance $\mathcal{K}_p(s_1, s_2) = \int_{t_0}^{s_1} \int_{t_0}^{s_2} \Psi(\tau_1)\mathcal{K}_v(\tau_1, \tau_2)\Psi^{\dagger}(\tau_2)d\tau_1 d\tau_2$. The covariance function is parameterized by the sequence of airplane orientations thereby demonstrating the fundamental link between tracking unresolved targets and high-resolution recognition algorithms.

The more "diffuse" component of the prior is developed by assuming the Euler angles are fixed for small sampling intervals, giving a sequence of angles $\phi(1), \phi(2), \ldots, \phi(j) = \phi(s), s \in [j\Delta, (j+1)\Delta)$. Then, the marginal on $\phi(j)$ takes the form of a Markov Von-Mises prior on the torus $\mathcal{T}(3)$ (see e.g. [15]) with the density $\prod_{i=1}^{3} \frac{1}{2\pi I_0(\kappa_i)} e^{\kappa_i cos(\phi_i(j) - \bar{\phi}_i(j))}$, where $I_0(\cdot)$ is the modified Bessel function of the first kind and order zero, and $\kappa = [\kappa_1 \ \kappa_2 \ \kappa_3]$ is the vector of concentration parameters, and $\bar{\phi(j)} = [\bar{\phi}_1(j) \ \bar{\phi}_2(j) \ \bar{\phi}_3(j)]$ is the mean of $\phi(j)$. The orientation process is made Markov by assigning the previous state as the mean of the present state, giving a potential of the form

$$\sum_{j} \sum_{i=1}^{3} \kappa_i cos(\phi_i(j) - \phi_i(j-1)) \, . \tag{8}$$

Recognition: We want to drive the algorithm towards deductions which are as simple as possible. Therefore, we use priors based on run-length coding to encourage hypotheses with minimal numbers of aggregated tracks. For this, associate with a target appearing at time $t_1^{(m)}$ and exiting at time $t_1^{(m)} + t_2^{(m)}$ the number of bits $log^* t_1^{(m)} + log^* t_2^{(m)} + log|\mathcal{A}| + \frac{6}{2} t_2^{(m)} log(sample-size)$, log^* the iterated logarithm $log + loglog + \ldots$ defined by Rissanen [16, 17] for constructing priors on the reals. Then the *complexity* prior for an M-track scene has potential

$$\left(log^* t_1^{(m)} + log^* t_2^{(m)} + log|\mathcal{A}| + \frac{6}{2} t_2^{(m)} log(sample - size) \right) \, . \tag{9}$$

3.2 Data Likelihood

The likelihood of the collected data correspond to two sensor types, a *tracking sensor* consisting of an array of passive sensors and a range radar, and a *high-resolution imaging* sensor.

Low resolution tracking: As shown in the left panel of Figure 2, for azimuth-elevation coordinate tracking, a cross array of n isotropic sensors is assumed as in [18, 19, 20, 21] using the standard narrowband signal model developed in [22]. Accordingly, depending on the geometry of the sensor arrangement, the phase lags of the signal reaching different sensor elements are known functions of the source locations. The deterministic signal model for sensor arrays is used in which the $n \times 1$ sensor array measurement vector $y_1(\tau)$, $\tau \in [t_0, t]$ is complex Gaussian distributed with diagonal covariance and mean $E\{y_1(\tau)\} = \sum_{m=1}^{\infty} d(x^{(m)}(\tau)) 1_{\mathcal{X}_0}(x^{(m)}(\tau)) s^{(m)}(\tau)$, with $s^{(m)}(\tau)$ the signal amplitude of the m-th track at time τ. Notice the indicator function $1_{\mathcal{X}_0}(x^{(m)}(\tau))$ selects the targets that contribute to the array manifold at time τ, $d(p^{(m)}(\tau))$ is the direction vector determined by the array geometry and the position of the m-th target.

Since the tracking array responds to the inertial positions of the target most naturally in azimuth and elevation, we convert from rectangular coordinates $p(t) = [p_1(t)\ p_2(t)\ p_3(t)] \in R^3$ to polar coordinates, $[r(t)\ \alpha_1(t)\ \alpha_2(t)] \in R^+ \times [0, 2\pi)^2$, (range, elevation and azimuth) using the standard relationship,

$$r = \sqrt{p_x^2 + p_y^2 + p_z^2}\ , \alpha_1 = arctan\frac{p_z}{\sqrt{p_x^2 + p_y^2}}\ , \alpha_2 = arctan\frac{p_y}{p_x}\ . \tag{10}$$

High resolution imaging: While the statistical models for high-resolution radar imaging are being incorporated in this problem by others ([23, 24, 25, 26, 27, 28]), all of the results shown here are based on an optical imaging system as depicted in the right panel of Figure 2. In this system, the data are a sequence of 2-D images resulting from projecting the scene volume containing targets onto the focal plane of the imaging sensor; i.e., the imaging process is modeled as a far field orthographic projection. Since the parameter set $x(M)$ completely determines the imaged volume, the projection is a deterministic operation from the parameter space to the measurement space $\Re^{\mathcal{L}\times[t_0,t]}$, $\mathcal{P} : \mathcal{X}_t \rightarrow \Re^{\mathcal{L}\times[t_0,t]}$, where $\mathcal{L}$ is the 2-D image space. For all of the results shown here the high-resolution imaging data is a non-zero mean white Gaussian process with mean the projective transformation of the scene: $E\{y_2(\tau)\} = \mathcal{P}(x(\tau))$, $\tau \in [t_0, t]$.

The posterior distribution is obtained as the product of the data likelihood and the prior density and is defined explicitly by Eqns. 11,12 below.

Remark: For observing the range locations of the targets, a range radar is assumed with the observations modeled as normally distributed with mean $|p(\tau)|$, the 2-norm of the position vector at time $\tau \in [t_0, t]$.

4 Random Sampling for Generating Conditional Expectations

4.1 The parameter space.

In particularizing the jump-diffusion algorithm to the multiple tracking problem it will be convenient to suppress explicit dependence on time t. Therefore, for each time t we will have a distribution for which the jump-diffusion process will be constructed. The parameter spaces themselves will be indexed by t, and form an increasing family of spaces.

The crucial part of the problem still remaining is the derivation of the inference algorithm: For all of the possible scenes we assume that the targets are stationary during some fundamental data sampling intervals, with the parameters and sensor data represented by their values on some index set $\{\tau_j\}_{j=1,\ldots,J}$, J the total number of sample points in the observation interval $\tau_j \in [t_0, t]$. Note, J is actually a function of t. Then an M-track parameter vector becomes $x(M) \in \mathcal{X}(M) \equiv (\mathcal{X}_0 \bigcup \{\emptyset\})^{MJ} \times \mathcal{A}^M$ with the complete parameter space being $\mathcal{X} = \bigcup_{M=0}^{\infty} \mathcal{X}(M)$. It will be useful to define the number of track segments in the m-th track $n^{(m)}$, and the total number of track-segments as $n(M)$, implying for example, $x(M) \in \mathcal{X}_0^{n(M)} \times \mathcal{A}^M$.

The **posterior** μ is of the Gibb's form with the potential H_M for $x(M) \in \mathcal{X}(M)$ becoming

$$\begin{aligned} H_M(x(M)) \;=\; & 1/\sigma_1 \sum_{j=1}^{J} |y_1(\tau_j) - \sum_{m=1}^{M} d(x^{(m)}(\tau_j)) 1_{\mathcal{X}_0}(x^{(m)}(\tau_j))|^2 + 1/\sigma_2 \sum_{j=1}^{J} |y_2(\tau_j) - \mathcal{P}(x(\tau_j))|^2 \\ & + \sum_{m=1}^{M} \left(p^{(m)} K_p^{-(m)} p^{(m)} + \sum_{j=2}^{n^{(m)}} \sum_{i=1}^{3} \kappa_i^{(m)} cos(\phi_i^{(m)}(\tau_j) - \phi_i^{(m)}(\tau_{j-1})) \right) \\ & + \sum_{m=1}^{M} \left(log^* n^{(m)} + log^* t_1^{(m)} + log|\mathcal{A}| + \frac{6}{2} n^{(m)} log(sample - size) \right) . \end{aligned} \tag{11}$$

The first two quadratic terms are associated with the tracking data and the high resolution imaging data. The last three terms are the prior terms on the tracking parameters, Von-Mises orientations and track complexity, respectively For arbitrary $x \in \mathcal{X}$, define the potential

$H_M(x) = 0$, for $x \notin \mathcal{X}(M)$. Then the posterior measure $\mu(\cdot)$ with density $\pi(\cdot)$ is in Gibb's form according to

$$\mu(dx) = \frac{\sum_{M=0}^{\infty} e^{-H_M(x)} 1_{\mathcal{X}(M)}(x)}{\mathcal{Z}} dx \ , \tag{12}$$

with the normalizer $\mathcal{Z} = \sum_{M=0}^{\infty} \int_{\mathcal{X}(M)} e^{-H_M(x)} dx$.

Now for the development which follows it will be convenient to define the part of the posterior which does not include the prior. This we define as L_M and is given by the formula

$$L_M(x(M)) = 1/\sigma_1 \sum_{j=1}^{J} |y_1(\tau_j) - \sum_{m=1}^{M} d(x^{(m)}(\tau_j)) 1_{\mathcal{X}_0}(x^{(m)}(\tau_j))|^2 + 1/\sigma_2 \sum_{j=1}^{J} |y_2(\tau_j) - \mathcal{P}(x(\tau_j))|^2 \ ,$$

with the prior potential term denoted by

$$\begin{aligned} P_M(x(M)) \ &= \ \sum_{m=1}^{M} \left(p^{(m)} K_p^{(m)} p^{(m)} + \sum_{j=2}^{n^{(m)}} \sum_{i=1}^{3} \kappa_i^{(m)} cos(\phi_i^{(m)}(\tau_j) - \phi_i^{(m)}(\tau_{j-1})) \right) \\ &+ \sum_{m=1}^{M} \left(log^* n^{(m)} + log^* t_1^{(m)} + log|\mathcal{A}| + \frac{6}{2} t_2^{(m)} log(sample - size) \right) \ . \end{aligned}$$

Remark: Notice, in identifying model M with an M-track configuration the potential must be adjusted so that the m-th track covariance $K_p^{(m)}$ is an $n^{(m)} \times n^{(m)}$ matrix appropriate for the m-th track. We would be more precise by identifying models $(n^{(1)}, n^{(2)}, \ldots n^{(M)}, M)$ with index $k \in \aleph$, from which the potential is then uniquely defined in the usual sense. However, with the subtle breach of notational convention we simply define H_M as the potential associated with an M-track configuration and modify the potential according to the variable numbers of parameters associated with the tracks.

4.2 The basic jump-diffusion set-up.

The crucial part of the problem still remaining is the derivation of the inference algorithm: that is *how shall we carry out hypothesis formation?* We follow the analysis outlined in [4, 1], in which conditional expectations with respect to the posterior density $\pi(\cdot)$ are generated empirically. First identify with each model an index $k \in \aleph$ with parameter space $\mathcal{X}(k)$ of dimension $n(k)$. The full hypothesis space $\mathcal{X} = \cup_{k=0}^{\infty}\mathcal{X}(k)$. The posterior distribution μ is then of the Gibb's type supported on $\mathcal{X}$, i.e. for all set $\mathcal{A} \subset \mathcal{X}$ Lebesgue measurable,

$$\begin{aligned} \mu(\mathcal{A}) &= \sum_{k=0}^{\infty} \mu(\mathcal{A} \cap \mathcal{X}(k)) , \\ &= \sum_{k=0}^{\infty} \int_{\mathcal{A}\cap\mathcal{X}(k)} \frac{e^{-H_k(x(k))}}{\mathcal{Z}} dx(k) . \end{aligned} \tag{13}$$

The goal is to essentially sample from μ generating a sequence of samples $X(s_1), X(s_2), \ldots$ with the property that

$$1/n \sum_{j=1}^{n} f(X(s_j)) \overset{n\to\infty}{\rightarrow} \int_{\mathcal{X}} f(x)\mu(dx) . \tag{14}$$

This we do via the construction of a Markov process $X(s)$ which satisfies *jump-diffusion dynamics* through $\mathcal{X}$ in the sense that (i) on random exponential times the process jumps from one of the countably infinite set of spaces $\mathcal{X}(k), k = 0, 1, \ldots$ to another, and (ii) between jumps it satisfies diffusions of dimension $n(k)$ appropriate for that space.

The process $X(s)$ within each of the multiple sub-spaces is a diffusion with infinitesimal drifts $a(x(k)) \in R^{n(k)}$ and infinitesimal variance matrix $B(x(k))$, an $n(k) \times n(k)$ matrix. It is the existence of this multiple disconnected union of spaces $\mathcal{X}$ which motivates the introduction of the second transformation type on the models, transformations which act by changing one model type to another with its resulting configuration. The transformations we shall term *simple moves* which are drawn probabilistically from a family $\mathcal{F}$ of changes in the model types $k \in \aleph$, and are applied discontinuously, with the simple moves defining transitions through $\aleph$, $\mathcal{F} : \aleph \rightarrow \aleph$. The family of transitions are chosen large enough to act transitively in the sense that given any pair $k', k'' \in \aleph$ it should be possible to find a finite chain of transitions that leads from k' to k''.

The set $\mathcal{F}$ controls the jump dynamics in the jump-diffusion processes as follows. The jump process corresponds to movement from one subspace to another on the jump times with transition probability measure $Q(x, dy) = \frac{q(x,dy)}{q(x)}$, $\int_{\mathcal{X}} Q(x, dy) = 1$. The measures $q(x, dy)$ are defined in the standard way [29]: $q(x, dy) = \lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon}\left(\Pr\{X(s+\epsilon) \in dy | X(s) = x\} - 1_{dy}(x)\right)$, with $q(x) = \int_{\mathcal{X} \setminus x} q(x, dy)$. The set $\mathcal{F}$ determines which measures are non-zero.

As we have shown for purely Euclidean spaces [4, 1], the proper choice of jump transition measures and diffusion drifts and variances (stochastic gradients) will make μ on $\mathcal{X}$ invariant with ergodic averages generated from the process converging to their expectations. See theorem 1 of [1].

4.2.1 The jump process.

The **jump process** is controlled by the family of changes $\mathcal{F}$ which control the movement through the non-connected subspaces. Changes in model type will include increasing and decreasing track length, increasing and decreasing number of tracks, and changing the target types. The set of transformations are defined as $\mathcal{F} = \{\vartheta^d_{t(j)}, \vartheta^d_{s(j)}, \vartheta^b_{t(j)}, \vartheta^b_{s(j)}, \vartheta_{a(j)}\}$. The first two correspond to deletion or removal of the j-th track and segment, which are mappings $\vartheta^d_{t(j)} : \mathcal{X}_0^{n(M)} \times \mathcal{A}^M \rightarrow \mathcal{X}_0^{n(M)-1} \times \mathcal{A}^{M-1}$, $\vartheta^d_{s(j)} : \mathcal{X}_0^{n(M)} \times \mathcal{A}^M \rightarrow \mathcal{X}_0^{n(M)-1} \times \mathcal{A}^M$, respectively. The second two are birth operators birthing tracks and segments to the j-th place or track, and are mappings $\vartheta^b_{t(j)} : \mathcal{X}_0^{n(M)} \times \mathcal{A}^M \rightarrow \mathcal{X}_0^{n(M)+1} \times \mathcal{A}^{M+1}$, $\vartheta^b_{s(j)} : \mathcal{X}_0^{n(M)} \times \mathcal{A}^M \rightarrow \mathcal{X}_0^{n(M)+1} \times \mathcal{A}^M$, respectively. The last operator simply changes the target type, $\vartheta_{a(j)} : \mathcal{X}_0^{n(M)} \times \mathcal{A}^M \rightarrow \mathcal{X}_0^{n(M)} \times \mathcal{A}^M$. It should be noted that the addition of only unit length tracks is allowed, and unit length track segments, as well as deletions of only unit length tracks or segments. Define the set, of indices of tracks in $x(M)$ which are candidates for deletion, by $\{m : 1 \leq m \leq M, n^{(m)} = 1\}$ and let $M_1(x(M))$ be the cardinality of this set. For the increments in parameter space, an explicit notation denoting a specific segment or track added to the configuration will be needed. Let $\oplus_j$ stand for the addition of track segments or tracks to the existing configuration, i.e. $x(M) \oplus_j y(1)$ represents an $M+1$ track configuration formed by adding $y(1)$ to $x(M)$ at the j-th location in the list, and $x(M) \oplus_j y$ signifies addition of a segment to the j-th track of $x(M)$.

These are the only transformations of model type that are allowed. To carry the evolution of the state forward from the diffusion we make the jump measures singular with respect to the Lebesgue measures in the respective subspaces which the jump transformations move into. For this, the part of the state which is not being added or deleted remains unchanged after the jump transformation. This corresponds to the following transition measures of the type,

$$\begin{aligned}
q(x(M), dy(M+1)) &= \sum_{j=1}^{M+1} q_t^b(x(M), y(M+1))\delta_{x(M)}\Big(d(\vartheta^d_{t(j)} y(M+1))\Big)dy(1) \,, \\
q(x(M), dy(M)) &= \sum_{j=1}^{M} q_s^b(x(M), y(M))\delta_{x(M)}\Big(d\Big(\vartheta^d_{s(j)} y(M)\Big)\Big)dy \\
&+ \sum_{j=1}^{M} q_s^d(x(M), y(M))\delta_{\vartheta^d_{s(j)} x(M)}(dy(M)) \,, \\
&+ \sum_{j=1}^{M} q_a(x(M), y(M))\delta_{x(M)}\Big(d\Big(\vartheta_{a(j)} y(M)\Big)\Big) \,, \\
q(x(M), dy(M-1)) &= \sum_{j=1}^{M} q_t^d(x(M), y(M-1))\delta_{\vartheta^d_{t(j)} x(M)}(dy(M-1)) \qquad (15)
\end{aligned}$$

Let $\mathcal{F}^1(x(M)) \subset \aleph$ be the set of models that can be reached from $x(M)$ in one jump move, and $\mathcal{X}(\mathcal{F}^1(x(M)))$ the space containing the configurations of these types. The total jump intensity becomes

$$q(x(M)) = \int_{\mathcal{X}(\mathcal{F}^1(x(M)))} q(x(M), dy) . \qquad (16)$$

4.2.2 The diffusion process.

The **diffusion process** between jumps controls the dynamics of $X(s)$ in their respective sub-spaces. For the sub-space associated with M tracks having $n(M)$ segments the diffusion flows through the manifold $\mathcal{X}_0^{n(M)} = \mathcal{T}(3)^{n(M)} \times \mathcal{R}^{3n(M)}$ associated with the orientations $\phi \in \mathcal{T}(3)$ and positions $p \in \mathcal{R}^3$. The restriction of a previous result in Theorem 1 from [1] to Euclidean spaces unfortunately prevents its direct application to the tracking problem in which the torus is involved. Even the most innocuous appearing stochastic differential equation (S.D.E.) make little sense when the manifold is curved in any way. This forces us to use more general results on Lie manifolds as described in [30] and adapted to the tracking case as follows.

Associate with the first $3n(M)$ components of the state vector the flow through $\mathcal{T}(3)^{n(M)}$, and the last $3n(M)$ components the flow through $\mathcal{R}^{3n(M)}$ according to $X(s) = [X_1(s), X_2(s)]$, $X_1(s) \in \mathcal{T}(3)^{n(M)}$, $X_2(s) \in \mathcal{R}^{3n(M)}$. Also, define $\frac{e^{-P(y(1)|x(M))}}{Z_{\mathcal{T}}(1)}$, $\frac{e^{-P_j(y|x(M))}}{Z_{\mathcal{S}}(1)}$ as the conditional prior densities on the single track and single segment spaces given the current configuration $x(M)$, respectively, with $Z_{\mathcal{T}}(1), Z_{\mathcal{S}}(1)$ being their normalizers. P_j denotes the attachment of the segment y to the j^{th}-track of the set $x(M)$. Then we have the following theorem.

Theorem 1 *IF the jump diffusion process $X(s)$ has the properties that*

1. *the diffusion $X(s)$ within any of the subspaces $\mathcal{X}_0^{n(M)}$ satisfies the S.D.E.*

$$X_1(s) = \left[X_1(0) + \int_0^s -\frac{1}{2}\nabla_1 H_M(X(\tau))d\tau + W_1(s)\right]_{\textbf{mod } 2\pi} , \qquad (17)$$

$$X_2(s) = X_2(0) + \int_0^s -\frac{1}{2}\nabla_2 H_M(X(\tau))d\tau + W_2(s) , \qquad (18)$$

where $[\cdot]_{\text{mod } 2\pi}$ *is taken componentwise,* ∇_1, ∇_2 *are the gradients with respect to the orientation and position (velocity) vectors respectively, and* $W_1(s), W_2(s)$ *are standard vector Wiener processes of dimensions* $3n(M)$*,*

2. *and the birth/death parameters of the jump measures*

$$\begin{aligned}
q_t^b(x(M), x(M)\oplus_j y(1)) &= \frac{1}{5(M+1)} e^{-[L_{M+1}(x(M)\oplus_j y(1)) - L_M(x(M))]_+} \frac{e^{-P(y(1)|x(M))}}{Z_T(1)}, \quad j = 1, .., M+1 \; ; \\
q_s^b(x(M), x(M)\oplus_j y) &= \frac{1}{5M} e^{-[L_M(x(M)\oplus_j y) - L_M(x(M))]_+} \frac{e^{-P_j(y|x(M))}}{Z_S(1)}, \quad j = 1, .., M \; , \\
q_t^d(x(M), \vartheta^d_{t(j)} x(M)) &= \frac{1}{5M_1(x(M))} \frac{e^{-[L_{M-1}(\vartheta^d_{t(j)} x(M)) - L_M(x(M))]_+}}{Z_T(1)} 1_{\{m>0\}}(M_1(x(M)) \; , \\
&\qquad j \in \{m : 1 \le m \le M, n^{(m)} = 1\} \; , \\
q_s^d(x(M), \vartheta^d_{s(j)} x(M)) &= \frac{1}{5M} \frac{e^{-[L_M(\vartheta^d_{s(j)} x(M)) - L_M(x(M))]_+}}{Z_S(1)} \; , \quad j = 1, .., M \; , \\
q_a(x(M), \vartheta_{a(j)} x(M)) &= \frac{1}{5M} e^{-[L_M(\vartheta_{a(j)} x(M)) - L_M(x(M))]_+}, \quad j = 1, .., M \; .
\end{aligned} \tag{19}$$

THEN, $X(s_j)$ *converges in variation norm to* μ*.*

Proof: The proof follows the general approach in [1, 30] with details summarized for this problem in the Appendix.

4.3 Algorithm Implementation

The jump-diffusion process satisfying Theorem 1 is constructed as follows. Initialize with $t_0 = 0, i = 0$.

1. Generate an exponential random variable u with mean 1.

2. For $s \in [s_i, s_i + u)$, $X(s)$ follows the stochastic differential Eqns. 17,18 in subspace determined by $X(s_i)$.

3. On random time $s_{i+1} = s_i + u$, define $x_{\text{old}} = X(s_{i+1}^-)$ and determine $M = M_{\text{old}}$ of x_{old} the number of tracks in $X(s_{i+1}^-)$.

4. Draw one of the 5 possible jump choices from the set $\{t^b, s^b, t^d, s^d, a\}$ according to the distribution $\{\frac{1/5}{Z}, \frac{1/5}{Z}, \frac{1/5 Z_T(1)}{Z} 1_{\{m>0\}}(M_1(x_{old})), \frac{1/5 Z_S(1)}{Z}, \frac{1/5}{Z}\}$, with $Z = \frac{1}{5} + \frac{1}{5} Z_T(1) 1_{\{m>0\}}(M_1(x_{old})) + \frac{1}{5} Z_S(1) + \frac{1}{5} + \frac{1}{5}$.

   If, t^b, then draw a 1-length track $y(1)$ from a uniform prior on $\mathcal{X}_0 \times \mathcal{A}$ and draw $j \in \{1, 2, \ldots, M+1\}$ uniformly:

$$x_{\text{new}} \leftarrow x_{\text{old}} \oplus_j y(1) \ .$$

   Else If, t^d, draw $j \in \{m : 1 \le m \le M, n^{(m)} = 1\}$ uniformly:

$$x_{new} \leftarrow \vartheta^d_{t(j)} x_{\text{old}} \ .$$

   Else If, s^b, then draw $j \in \{1, 2, \ldots, M\}$ uniformly and draw $y \in \mathcal{X}_0$ from the Von-Mises prior on $\mathcal{T}(3)$ and the Gaussian prior on $\Re^3$, $\frac{e^{-P_j(y|x(M))}}{Z_S(1)}$, conditioned on the current j^{th} track configuration:

$$x_{new} \leftarrow x_{\text{old}} \oplus_j y \ .$$

   Else If, s^d, draw $j \in \{1, 2, \ldots, M\}$ uniformly:

$$x_{new} \leftarrow \vartheta^d_{s(j)} x_{\text{old}} \ .$$

   Else, Draw $j \in \{1, 2, \ldots, M\}$ uniformly:

$$x_{new} \leftarrow \vartheta_{a(j)} x_{\text{old}} \ .$$

5. Determine M_{new} of x_{new}.

6. If, $L_{M_{\text{old}}}(x_{\text{old}})) - L_{M_{\text{new}}}(x_{\text{new}}) > 0$, $X(s_{i+1}) \leftarrow x_{\text{new}}$

   Else $X(s_{i+1}) \leftarrow x_{\text{new}}$ with probability $e^{-[L_{M_{\text{new}}}(x_{\text{new}}) - L_{M_{\text{old}}}(x_{\text{old}})]}$ and

   Else $X(s_{i+1}) \leftarrow x_{\text{old}}$ with probability $1 - e^{-[L_{M_{\text{new}}}(x_{\text{new}}) - L_{M_{\text{old}}}(x_{\text{old}})]}$.

7. $i \leftarrow i+1$, return to 1.

Since a track-segment of length 1 correspond to $y \in \mathcal{X}_0$ consisting of the position and orientation components, the discretized form of Eqn. 4 is used for the position and the Markov Von-Mises prior on the torus $\mathcal{T}(3)$ used for the orientation component. The candidates for deletion are obtained by removing the last segment from the current j^{th} track estimate or the j^{th} track estimate itself.

5 Results

Below we focus on single track identification in 3-D space; see [31] for multiple target tracking in 2-D.

For the implementation of the jump-diffusion algorithm for estimating the motion of a single target, i.e. M = 1, the parameter space becomes $\mathcal{X}_0^{[t_0,t]} \times \mathcal{A}$. For the implementation there are a total of two target types, $\mathcal{A} \equiv \{1,2\}$. The algorithm was jointly implemented using the *flight simulator* software on the Silicon Graphics workstation for generating the data sets, and a massively parallel 4096 processor SIMD DECmpp/MasPar machine for implementing the tracking-recognition algorithm. Figure 4 shows the simulation environment for a sample target-flight observed by sensor systems located on ground represented by the mesh. The target motion is observed at 1500 times during the flight.

The array geometry corresponds to a 64-element cross-array of isotropic sensors located at half-wavelength spacing. The tracking data $\{y_1(\tau), \tau \in [t_0, t)\}$ is a 64-element complex vector with mean $d(p(\tau))s(\tau)$ and additive complex Gaussian white noise of the Goodman's class. The direction vector corresponds to the array takes the form

$$[e^{-\frac{i31}{2}\lambda_1(\tau)},\ e^{-\frac{i29}{2}\lambda_1(\tau)}\ ,\ldots, e^{\frac{i31}{2}\lambda_1(\tau)},\ e^{-\frac{i31}{2}\lambda_2(\tau)}, e^{-\frac{i29}{2}\lambda_2(\tau)}\ ,\ldots,\ e^{\frac{i31}{2}\lambda_2(\tau)}]^T\ ,\quad i = \sqrt{-1}\ ,$$

where $\lambda_1(\tau) = \pi cos(\alpha_1(\tau))sin(\alpha_2(\tau))$, and $\lambda_2(\tau) = \pi cos(\alpha_1(\tau))sin(\alpha_2(\tau))$, $\alpha_1(\tau), \alpha_2(\tau)$ is the azimuth, elevation angles of the target position $p(\tau)$. Since we use the velocity representation the azimuth and elevations are generated using the standard coordinate transformation of Eqns. 5,10. The upper panels in Figure 3 display the azimuth-elevation power spectra of the tracking data, generated by projecting the data vector onto the candidate direction vectors, for two target locations.

The 2-D imaging data $\{y_2(\tau), \tau \in [t_0, t)\}$ consists of 4096 Gaussian random variables associated with a 64×64 imaging lattice. The mean is $\mathcal{P}(x(t))$ where $\mathcal{P}(\cdot)$ is simply the 2-D projection of the rendered object positioned and oriented at $p(\tau), \phi(\tau)$, with additive noise. The lower panels in Figure 3 show two data samples obtained by high resolution imaging of the target along its flight.

At any given time t the jump-diffusion algorithm is run to generate samples from the posterior distribution generated by the data up to time t. This simulation is performed until the next data set arrives at $t+1$ when the algorithm starts sampling from the new posterior. For sampling the jump-diffusion Markov process is constructed as follows. For the single object case the possible jump transformations through parameter space involve either addition of a track segment $y \in \mathcal{X}_0$, deletion of a track segment, or a change of target type. The set of changes

$\mathcal{F} \equiv \{\vartheta^d_{s(1)}, \vartheta^b_{s(1)}, \vartheta_{a(1)}\}$ are transformations of the type

$$x(1) \in \mathcal{X}_0^{n(1)} \times \mathcal{A} \;\rightarrow\; x(1) \oplus_1 y \in \mathcal{X}_0^{n(1)+1} \times \mathcal{A} \,, \tag{20}$$

$$x(1) \in \mathcal{X}_0^{n(1)} \times \mathcal{A} \;\rightarrow\; \vartheta^d_{s(1)} x(1) \in \mathcal{X}_0^{n(1)-1} \times \mathcal{A} \,, \tag{21}$$

$$x(1) \in \mathcal{X}_0^{n(1)} \times \mathcal{A} \;\rightarrow\; \vartheta_{a(1)} x(1) \in \mathcal{X}_0^{n(1)} \times \mathcal{A} \,. \tag{22}$$

Shown in Figure 4 is the evolution of the random sampling algorithm for estimating the target track. The grey track represents the true airplane path, consisting of 1500 track segments, used in data generation with the estimated track shown overlapping in black at three different times during the estimation. Figure 5 shows a magnified view of a section of the track, formed of 8 track-segments, being estimated by the jump-diffusion algorithm. The top 4 panels illustrate the jump part of the algorithm for which we have turned off the diffusion. These upper panels shows successive guesses of the jump process which continually attempt to add and delete new track segments. Since the actual object has created a path which is longer then that which has been inferred by the algorithm during the early segments, the jump process always chooses to add new track segments. Notice, that on each addition the new segment is drawn from the prior on flight dynamics, which are parameterized by the track up to that point in time. Hence, the jump algorithm tends to infer track segments which are close to the true track if the current state vector is close to it. Because the diffusion has been turned off, notice the disparity between the track and the state of the algorithm.

The lower panels show the result of applying the diffusion to the state vector. The flow of the panels corresponds to increasing simulation time as the diffusion simulates from the posterior with the state brought into alignment.

Figure 6 depicts the importance of the dynamics based prior. Based on the equations of motion and the track history the candidate segments are generated and accepted/rejected according to their likelihood. To show the support of the prior distribution in "phase space", the upper panels plot the 10 highest prior probability candidates placed at the track end for the algorithm to choose from. Each panel corresponds to a different time during the inference. The top row shows that if the track vector is close to the true track, the cone of candidates predicts well the future position. The lower panel shows the effect of the track state deviating from the true path, where the cone of prediction is not close to the future airplane position.

Figure 7 demonstrates the global importance of the prior distribution in estimating a portion of the target path. The algorithm was run with and without the prior measure, under the same parameters, with the results shown in the figure. The upper panels show the sequence of estimates obtained from the algorithm without any information from airplane dynamics. The lower panels use the prior information based on the equations of motion describing the airplane flight.

6 Acknowledgments

We would like to thank Robert Teichman for a great deal of help in the recognition component of the algorithm. We owe special thanks to Professor Yali Amit who provided the detailed development for the diffusion on Lie groups of which the torus is a particular example.

7 Appendix

Proof of Theorem 2: The proof has two parts: (i) showing that π is an invariant density of the process, and (ii) verifying that the process is irreducible and therefore π is the unique invariant density. Part (ii) follows directly that in [30] using the properties of the jump process and the fact that the diffusions are each irreducible over their respective subspaces. In part (i) we need to verify the stationarity for both the jump and diffusion components of the Markov process. The generator, or backward Kolmogoroff operator, for the jump-diffusion process (denote it as $A = A^d + A^j$ (diffusion+jump)) characterizes the stationary density in that $\pi(x)$ is stationary for the jump-diffusion if and only if

$$\int Af(x)\pi(x)dx = 0 \tag{23}$$

for all f in the domain of A, $\mathcal{D}(A)$.

The diffusion process has two components corresponding to the S.D.E. on the multi-dimensional torus (17) and the S.D.E. on the Euclidean space of target positions (velocities) (18). To prove invariance of $\pi(x)$ for the diffusion on the torus we use results from [30] on invariant distributions of S.D.E.'s on manifolds, in particular the multi-dimensional torus. To demonstrate the approach, we prove the stationarity condition for the Euclidean component only. Define a set of functions which forms the domain of the generator A as

$$\mathcal{D}(A) = \{f : f = \sum_{M=0}^{K} 1_{\mathcal{X}(M)} f_M, f_M \in \hat{C}^2(\mathcal{X}(M)), K \geq 0\} , \tag{24}$$

with $\hat{C}^2$ twice continuously differentiable functions vanishing at ∞. Then the infinitesimal generator for diffusion A^d acting on such a function, $f = \sum_{M=0}^{K} 1_{\mathcal{X}(M)} f_M$, according to Eqn. 23 gives

$$\begin{aligned}\int_{\mathcal{X}} A^d f(x)\pi(x)dx &= \sum_{M=0}^{K} [- \int_{\mathcal{X}(M)} \frac{1}{2} < \nabla H_M(x), \nabla f_M(x) > \frac{e^{-H_M(x)}}{\mathcal{Z}} dx \\ &\quad + \int_{\mathcal{X}(M)} \frac{1}{2} \left(\sum_{i=1}^{3n(M)} \frac{\partial^2 f_M(x)}{\partial p_i^2} \right) \frac{e^{-H_M(x))}}{\mathcal{Z}} dx \;].\end{aligned}$$

where $< \cdot, \cdot >$ stands for the vector dot-product and the gradients $\nabla H_M(x), \nabla f_M(x)$ are w.r.t the position (velocity) vector, an element of $R^{3n(M)}$. Integration by parts of the second term, with the fact that the functions f_M vanish at the boundary, results in a term which is negative of the first term. Therefore the given posterior π is the stationary density of the diffusion process.

We note that the curved nature of the torus requires the argument to be modified in sufficiently subtle ways. For details of such modifications to the manifolds associated with Lie groups see [30]. The jump part of the generator A^j is given by

$$A^j f(x) = q(x) \int_{\mathcal{X}(\mathcal{F}^1(x))} Q(x, dy)(f(y) - f(x)) \; ,$$

and computing the adjoint corresponding to the Eqn. 23 (see [1] for illustration) provdes the balance condition

$$q(x)\pi(x)dx = \int_{\mathcal{X}(\mathcal{F}^{-1}(x))} q(y, dx)\pi(y)dy \; , \tag{25}$$

where $\mathcal{F}^{-1}(x) \subset \aleph$ is the subset of models which can reach x in one jump transition. The jump parameters must satisfy Eqn. 25 for the density $\pi(x)$ to be stationary for the jump part of the process. For definiteness, assume $x \in \mathcal{X}(M)$ so that $x \equiv x(M)$ is an M-track configuration. Substituting for the transition measures from Eqns. 15 gives

$$\begin{aligned}
\pi(x(M))dx(M) \quad & [\quad \sum_{j=1}^{M} \int_{\mathcal{X}_0} q_s^b(x(M), x(M) \oplus_j y)dy + \sum_{j=1}^{M} q_s^d(x(M), \vartheta_{s(j)} x(M)) \\
& + \sum_{j=1}^{M+1} \int_{\mathcal{X}_0 \times \mathcal{A}} q_t^b(x(M), x(M) \oplus_j y(1))dy(1) + \sum_{j=1}^{M} q_t^d(x(M), \vartheta_{t(j)} x(M)) \\
& + \sum_{j=1}^{M} q_a(x(M), \vartheta_{a(j)} x(M)) \;]
\end{aligned}$$

$$= dx(M) \, [\sum_{j=1}^{M} \int_{\mathcal{X}_0} q_s^d(x(M) \oplus_j y, x(M))\pi(x(M) \oplus_j y)dy + \sum_{j=1}^{M} q_s^b(\vartheta_{s(j)} x(M), x(M))\pi(\vartheta_{s(j)} x(M))$$

$$+ \sum_{j=1}^{M+1} \int_{\mathcal{X}_0 \times \mathcal{A}} q_t^d(x(M) \oplus_j y(1), x(M))\pi(x(M) \oplus_j y(1))dy(1) + \sum_{j=1}^{M} q_t^b(\vartheta_{t(j)}x(M), x(M))\pi(\vartheta_{t(j)}x(M))$$
$$+ \sum_{j=1}^{M} q_a(\vartheta_{a(j)}x(M), x(M))\pi(\vartheta_{a(j)}x(M)) \;] \, .$$

We will prove this equality treating only the first two summation terms from both sides, corresponding to the birth/death of track-segments; the treatment for the rest being similar. The jump moves considered here, birth/death of track-segments, are defined by Eqns. 20 and 21. Substituting the values for q_s^b, q_s^d from Eqn. 19,

L.H.S.:

$$\begin{aligned}
&\pi(x(M)) \, \frac{dx(M)}{Z_S(1)} \, \frac{1}{5M} \sum_{j=1}^{M} [\int_{\mathcal{X}_0} e^{-[L_M(x(M)\oplus_j y) - L_M(x(M))]_+} e^{-P_j(y|x(M))} dy \\
&\qquad + e^{-[L_M(\vartheta_{s(j)}x(M)) - L_M(x(M))]_+} \;] \\
= \; & \frac{dx(M)}{(\mathcal{Z})(Z_S(1))} \, \frac{1}{5(M)} \sum_{j=1}^{M} [\int_{\Omega_>} e^{-L_M(x(M))} e^{-P_M(x(M))} e^{-P_j(y|x(M))} dy \\
&\qquad + \int_{\Omega_\leq} e^{-L_M(x(M)\oplus_j y)} e^{-P_M(x(M))} e^{-P_j(y|x(M))} dy \\
&\qquad + e^{-L_M(x(M))} e^{-P_M(x(M))} e^{-[L_M(\vartheta_{s(j)}x(M)) - L_M(x(M))]_+} \;] \, . \qquad (26)
\end{aligned}$$

R.H.S.:

$$\begin{aligned}
&\frac{dx(M)}{Z_S(1)} \, \frac{1}{5(M)} \sum_{j=1}^{M} [\int_{\mathcal{X}_0} e^{-[L_M(x(M)) - L_M(x(M)\ominus_j y)]_+} \pi(x(M) \oplus_j y) dy \\
&\qquad + e^{-[L_M(x(M)) - L_M(\vartheta_{s(j)}x(M))]_+} e^{-P_j(y|\vartheta_{s(j)}x(M))} \pi(\vartheta_{s(j)}x(M)) \;] \\
= \; & \frac{dx(M)}{(\mathcal{Z})(Z_T(1))} \, \frac{1}{5M} \sum_{j=1}^{M} [\int_{\Omega_>} e^{-L_M(x(M))} e^{-P_M(x(M)\oplus_j y)} dy \\
&\qquad + \int_{\Omega_\leq} e^{-L_M(x(M)\oplus_j y)} \; e^{-P_M(x(M)\oplus_j y)} dy \\
&\qquad + e^{-[L_M(x(M)) - L_M(\vartheta_{s(j)}x(M))]_+} e^{-P_j(y|\vartheta_{s(j)}x(M))} e^{-L_M(\vartheta_{s(j)}x(M))} e^{-P_M(\vartheta_{t(j)}x(M))} \;] \, , \; (27)
\end{aligned}$$

where

$$\Omega_{>} = \mathcal{X}_0 \bigcap \{y : L_M(x(M)) > L_M(x(M) \oplus_j y)\}$$

$$\Omega_{\leq} = \mathcal{X}_0 \bigcap \{y : L_M(x(M)) \leq L_M(x(M) \oplus_j y)\} .$$

Comparing Eqns. 26, 27 and combining the prior terms (i.e. $P_M(x(M) \oplus_j y) = P_M(x(M)) + P_j(y|x(M))$, and $P_M(x(M)) = P_M(\vartheta_{s(j)} x(M)) + P_j(y|\vartheta_{s(j)} x(M))$), the equality is verified.

Q.E.D

References

[1] U. Grenander and M. I. Miller. Representations of knowledge in complex systems. *Journal of the Royal Statistical Society B*, 56(3):549–603, 1994.

[2] B. Gidas. Metropolis type monte-carlo simulation algorithms and simulated annealing. In *Trends of Contemporary Probability*, 1993. to appear.

[3] J. Besag and P. J. Green. Spatial statistics and bayesian computation. *J. Royal Statistical Society B*, 55:25–38, 1993.

[4] U. Grenander and M. I. Miller. Jump-diffusion processes for abduction and recognition of biological shapes. *Monograph of the Electronic Signals and Systems Research Laboratory*, 1991.

[5] Y. Bar-Shalom and E. Tse. Tracking in cluttered environment with probabilistic data association. *Automatica*, (11):451–460, 1975.

[6] Y. Bar-Shalom and T. E. Fortmann. *Tracking and Data Association*. Academic Press, 1988.

[7] Editor: Y. Bar-Shalom. *Multitarget-Multisensor Tracking*. Artech House, 1990.

[8] C. R. Rao, C. R. Sastry, and B. Zhou. Tracking the direction of arrival of multiple moving targets. *IEEE Transactions on Acoustics,Speech and Signal Processing*, accepted for publication, 1993.

[9] C. R. Sastry, E. W. Kamen, and M. Simaan. An efficient algorithm for tracking angles of arrival of moving targets. *IEEE Transactions on Acoustics,Speech and Signal Processing*, ASSP-39(No.1):242–246, 1991.

[10] C. K. Sword, M. Simaan, and E. W. Kamen. Multiple target angle tracking using using sensor array outputs. *IEEE Trans. AES.*, 26(2):367–373, 1990.

[11] W. Freiberger and U. Grenander. Computer generated image algebras. *Internation Federation Information Processing*, 68:1397–1404, 1969.

[12] U. Grenander. Advances in pattern theory: The 1985 Rietz lecture. *The Annals of Statistics*, 17(1):1–30, 1989.

[13] Nicholas Cutaia and Joseph A. O'Sullivan. Automatic target recognition algorithms using kinematic priors. *Electronic Signals and Systems Research Lab. Monograph*, 1994.

[14] Bernard Friedland. *Control System Design : An Introduction To State-Space Methods.* McGraw-Hill Book Company, 1986.

[15] K.V. Mardia. *Statistics of Directional Data.* Academic Press, London and New York, 1972.

[16] J. Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 11:416–431, 1983.

[17] J. Rissanen. Stochastic complexity and modeling. *The Annals of Statistics*, 14, no.3:1080–1100, 1986.

[18] M.I. Miller and D. R. Fuhrmann. Maximum likelihood narrow-band direction finding and the EM algorithm. *IEEE Acoust. Speech and Signal Processing*, 38, No.9:560–577, 1990.

[19] A. Srivastava, M.I. Miller, and U. Grenander. Jump-diffusion processes for object tracking and direction finding. In *Proceedings of the 29th Annual Allerton Conference on Communication, Control and Computing*, pages 563–570, Urbana, Champaign, 1991. University of Illinois.

[20] A. Srivastava, N. Cutaia, M.I. Miller, J. A. O'Sullivan, and D. L. Snyder. Multi-target narrowband direction finding and tracking based on motion dynamics. In *Proceedings of the 30th Annual Allerton Conference on Communication, Control and Computing*, Urbana, Champaign, 1992. University of Illinois.

[21] M.I. Miller, R. S. Teichman, A. Srivastava, J.A. O'Sullivan, and D. L. Snyder. Jump-diffusion processes for automated tracking-target recognition. In *Proceedings of the Twenty-Seventh Annual Conference Conference on Information Sciences and Systems*, pages 617–622, Baltimore, Maryland, March 24-26 1993. Johns Hopkins University.

[22] R. Schmidt. *A signal subspace approach to multiple emitter location and spectral estimation.* Ph.D. Dissertation of Stanford University, Palo Alto, CA., Nov. 1981.

[23] D.L. Snyder, J.A. O'Sullivan, and M.I. Miller. The use of maximum-likelihood estimation for forming images of diffuse radar-targets. In *Transactions of SPIE in Advanced Architectures and Algorithms*, San Diego, California, 1987.

[24] D.L. Snyder, J.A. O'Sullivan, and M.I. Miller. The use of maximum-likelihood estimation for forming images of diffuse radar-targets from delay-doppler data. *IEEE Transactions on Information Theory*, 35(3):536–548, 1989.

[25] J.A. O'Sullivan, P. Moulin, and D.L. Snyder. Cramer-rao bounds for constrained spectrum estimation with application to a problem in radar imaging. In *Proceedings 26th Allerton Conference on Communication, Control, and Computing*, Champaigne, Urbana; October 1988. Urbana, IL.

[26] M.I. Miller, D.R. Fuhrmann, J.A. O'Sullivan, and D.L. Snyder. Maximum-likelihood methods for toeplitz covariance estimation and radar imaging. In Simon Haykin, editor, *Advances in Spectrum Estimation*, pages 145–172. Prentice-Hall, 1990.

[27] P. Moulin, J.A. O'Sullivan, and D.L. Snyder. A method of sieves for multiresolution spectrum estimation and radar imaging. *IEEE Transactions on Information Theory*, 1992.

[28] J.A. O'Sullivan, K. C. Du, R. S. Teichman, M.I. Miller, D.L. Snyder, and V.C. Vannicola. Radar target recognition using shape models. In *Proc. 30th Annual Allerton Conference on Communication,Control, and Computing*, pages 515–523, Urbana, Il., 1992. University of Illinois.

[29] I. I. Gihman and A. V. Skorohod. *Introduction to the Theory of Random Processes.* Saunders, Philadelphia, 1965.

[30] Y. Amit and M.I. Miller. Ergodic properties of jump-diffusion processes. *Monograph of the Electronic Signals and Systems Research Laboratory*, January 1993.

[31] A. Srivastava, M.I. Miller, and U. Grenander. Multi-target direction of arrival tracking. *IEEE Transactions on Signal Processing*, to appear, May 1995.

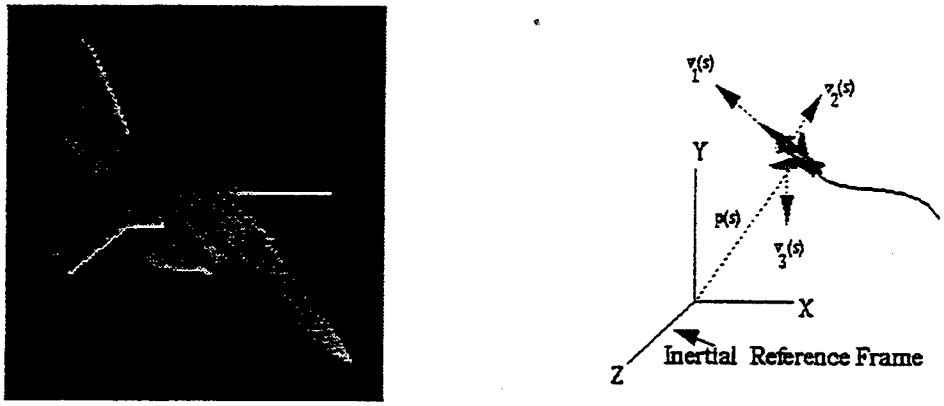


Figure 1: The left panel shows the 3-D target generator $g \in \mathcal{G}^o$ under a similarity transformations. The right panel shows the target located at position $p(s)$, oriented at $\phi(s)$ with velocities $v(s)$ resolved in the body frame coordinates.

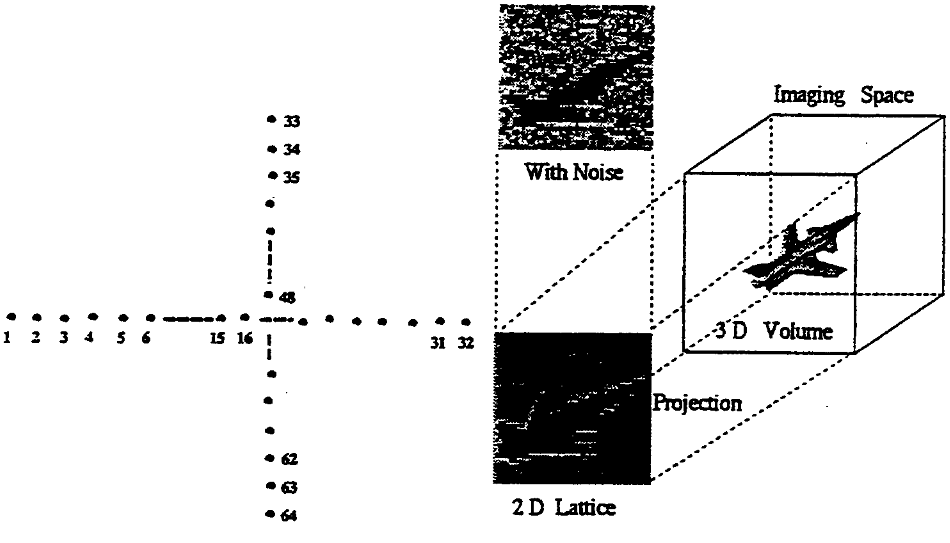


Figure 2: The left panel displays the cross array of isotropic sensors at half wavelength spacing, used to observe the angular location of the target. The right panel shows the far-field orthographic imaging system used for observing the targets at a high resolution.

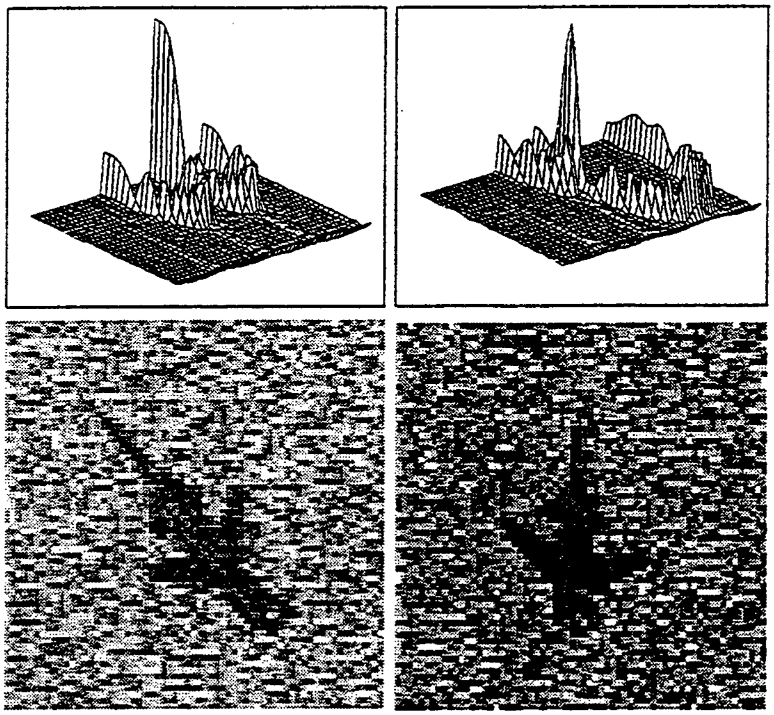


Figure 3: The upper panels show the azimuth-elevation power spectrum of the tracking data at two sample times. The lower panels display the high resolution data sets for the target at two different times during the flight path.

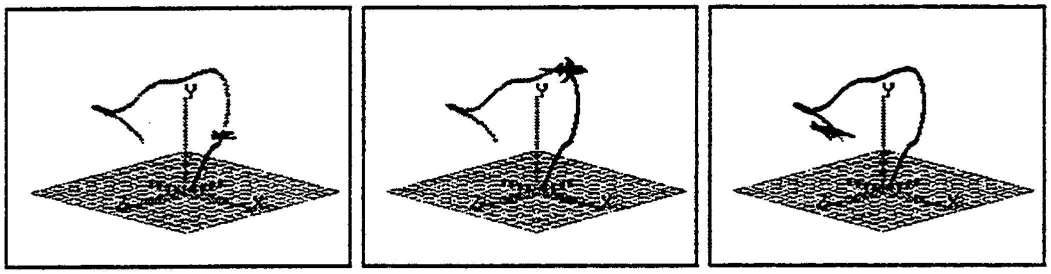


Figure 4: The actual track drawn in grey is observed by the ground based observation system. The track estimates are drawn overlapping in black at three stages of the algorithm.

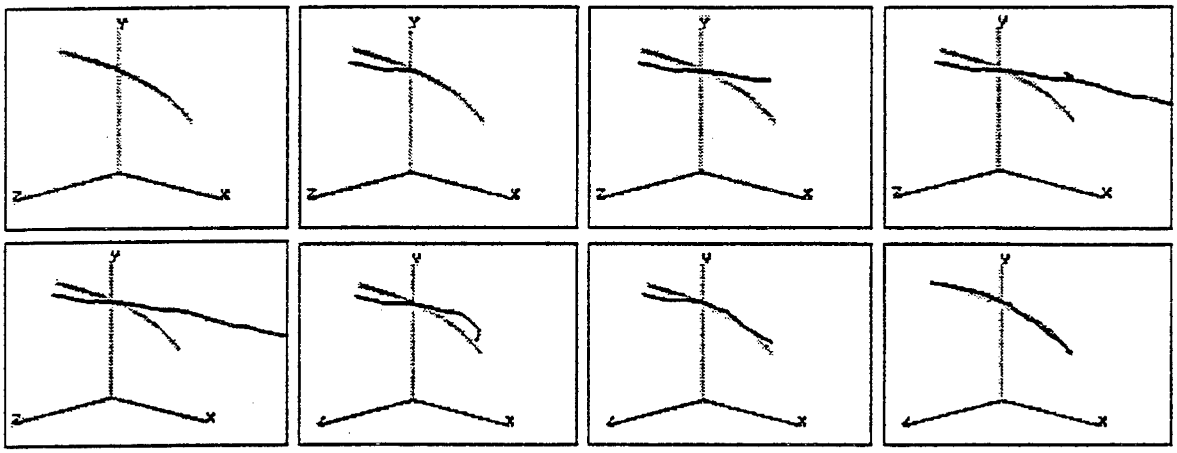


Figure 5: The upper panels show the sequence of jump moves adding segments to the estimated state from left to right, with the diffusion turned off. The lower panels show the continuous diffusion transformation aligning the estimated to the true track via the gradients on the posterior energy.

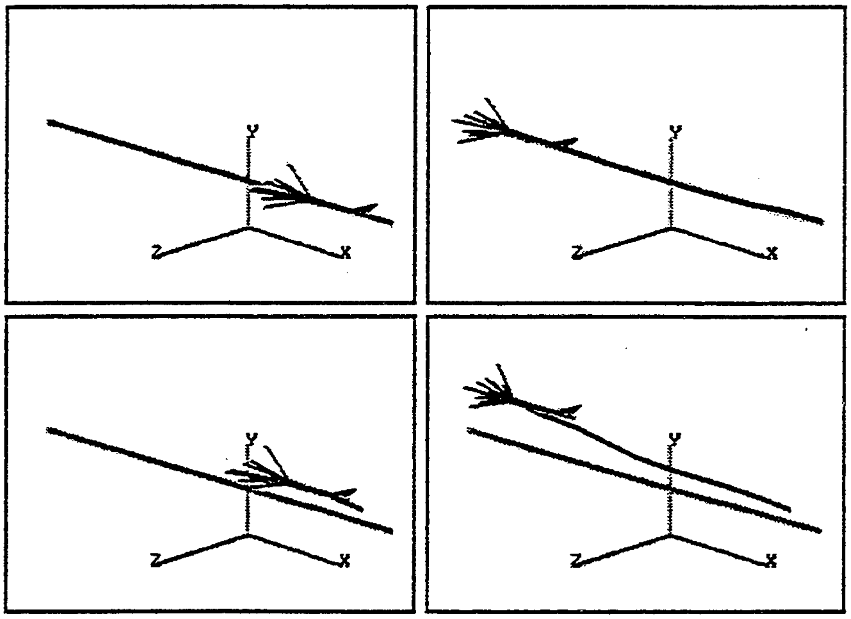


Figure 6: The four panels show candidates from the prior distribution for target path estimation with the high prior probability candidates forming a cone at track end for the algorithm to sample from. The upper panels show the prior with the diffusion on track parameters turned on; the lower panels have the diffusion turned off.

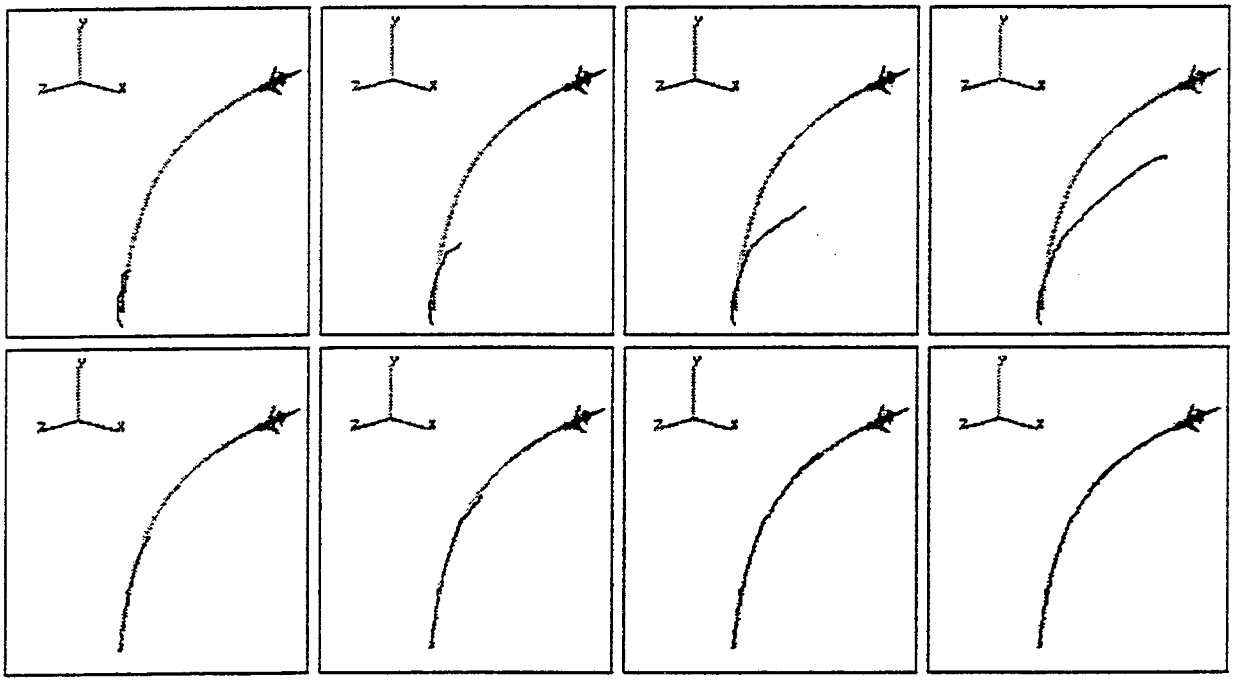


Figure 7: The upper panels display the estimated states at four times without the tracking prior information. The lower panels show the results of the dynamics based estimation algorithm with the prior included.

Appendix B

Table of Contents

A. Srivastava, R.S. Teichman, M. I. Miller, D. L. Snyder, and J. A. O'Sullivan, "Jump-Diffusion Based Sampling Algorithm for Target Tracking and Recognition," 27th Asilomar Conf. on Signals, Systems, and Computers, November 1-3, 1993, Pacific Grove, CA. . . . . pg B-2

J. A. O'Sullivan, S.P. Jacobs, M. I. Miller, and D. L. Snyder, "A Likelihood-Based Approach to Joint Target Tracking and Identification," 27th Asilomar Conf.on Signals, Systems, and Computers, November 1-3, 1993, Pacific Grove, CA. .pg B-7

J. A. O'Sullivan, K. C. Du, R. S. Teichman, M. I. Miller, D. L. Snyder and V. C. Vannicola, "Reflectivity Models for Radar Target Recognition," SPIE International Symposium on OE/Aerospace Sensing, Orlando, April 1993. . . . . . .pg B-12

J. A. O'Sullivan, M. I. Miller, A. Srivastava, and D. L. Snyder, "Tracking Using a Random Sampling Algorithm," 12th world Congress of the International Federation of Automatic Control, Sydney, Australia, July 1993. . . . . .pg B-23

M. I. Miller, R. Teichman, A. Srivastava, J. A. O'Sullivan, and D. L. Snyder, "Jump-Diffusion Processes for Automated Tracking-Target Recognition," Proceedings of the 27th Annual Conference on Information Sciences and Systems, Johns Hopkins University, Baltimore, March 1993, pp. 617-622. . . . pg B-27

A. Srivastava, R. S. Teichman, and M. I. Miller, "Target Tracking and Recognition Using Jump-Diffusion Processes," ARO's 11th Army Conf. on Applied Mathematics and Computing, June 8-11, 1993, Carnegie Mellon University, Pittsburgh. . . . . pg B-33

A. Srivastava, N. Cutaia, M. I. Miller, J. A. O'Sullivan, and D. L. Snyder, "Multi-Target Narrowband Direction Finding and Tracking Based on Motion Dynamics," Proceedings of the 30th Annula Allerton Conference on Communication, Control, and Computing, Urbana, IL, October 1992. . . . . .pg B-39

A. Srivastava, M. I. Miller, U. Grenander, "Lie Group Parameterization for Dynamics Based Prior in ATR," reprint. . . . . pg B-49

N. J. Cutaia and J. A. O'Sullivan, "Automatic Target Recognition Using Kinematic Priors, " Proceedings CDC, December 1994, ... pg B-51

Jump-Diffusion Based Sampling Algorithm For Target Tracking and Recognition

Anuj Srivastava Robert S. Teichman Michael I. Miller

Donald L. Snyder Joseph A. O'Sullivan

Electronic Signals & Systems Research Lab
Washington University
St Louis, MO 63130

Abstract

A new random sampling algorithm for recognition and tracking of an unknown number of targets and target types is presented. Taking a Bayesian approach we define a posterior measure on the parameter space by combining the observed data likelihood with a prior based on airplane dynamics. The Newtonian force equations governing the airplane motion are utilized to form the prior density on the airplane positions. The sampling algorithm based on Jump-Diffusion processes, first introduced by [Grenander,Miller] (1991), is derived for generating high probability estimates of target positions, orientations and types from the posterior measure. Results are presented from its joint implementation on the Silicon Graphics workstation and the DECmpp SIMD machine distributing data-simulation, visualization and computation over network.

1 Introduction

Automated target tracking and recognition are well known problems in signal processing and control system literature. They represent a class of problems which utilize time records of multiple sensors to estimate the charaterstics of arriving signals. A great deal of published work in the control literature on solving the non-linear tracking problem [1, 2, 3] involves the Kalman filter based techniques. This requires using some linear approximations valid in particular cases but the general non-linear problem remains unsolved. Recently, there has been considerable interest in tracking the directions of arrival (DOAs) by an array of passive sensors [4, 5, 6] using variations of the gradient-techniques in mostly maximum-likelihood settings. Again these approaches utilize simplifying assumptions like multiple snapshots at each target position, known, fixed number of targets and restricted target motion. Also, due to the efforts to solve the tracking and recognition problems separately the target-dynamics is not efficiently utilized.

We present a new random sampling based solution for the tracking and recognition problems in a general setting. The aim is to track-recognize an unknown number of non-cooperative/hostile targets which appear and disappear at random times. The targets are observed from multiple sensors: an azimuth-elevation tracking array and a high resolution radar for target recognition. For moving targets we model a single snapshot for each target position. The rotational and translational motions of the airplane are coupled to each other by a set of differential equations modeling the motion. It results in the dynamics based prior on airplane's positions being parameterized by its rotational parameters connecting the tracking and recognition together. We utilize the collected data up to any fixed time t to generate minimum-mean-square-error (*MMSE*) estimates of the parameters describing target paths from the initial time t_0 to t. Since *MMSE* estimate is the conditional mean of the parameter under posterior density, we perform a random sampling of the posterior density, based on Jump-Diffusion processes, to estimate these conditional means [7, 8]. The time samples of this jump-diffusion process visit the elements in the parameter space $\mathcal{X}_t$ according to the posterior density. More importantly, the samples generated by this Markov process can be utilized to empirically reconstruct the posterior distribution or to evaluate any mean associated with it.

2 Parameter Set

We use the global shape models and pattern theoretic approach introduced by Grenander [9] to analyze complex scenes. The observed scene at any time consists of a set of generators each representing a target at its position and orientation. The fundamental variabilities in these scenes are accommodated via the rotation and translation transforms on the templates which form the building blocks of the scene, as described on [10]. Define $\mathcal{M}(3) \equiv [0, 2\pi]^3$, with $0, 2\pi$ identified, as the three-dimensional torus and $\mathcal{A}$ the discrete alphabet of target types. Then $\vec{\phi} \in \mathcal{M}(3)$ is the triple of Euler angles (pitch, roll and yaw), $\vec{p} \in \Re^3$ is the position vector of the target, and $a \in \mathcal{A}$ is the target type. For the non-cooperative environments where the m^{th} object appears and disappears at random times $t_1^{(m)} + 1, t_1^{(m)} + t^{(m)}$ with its duration of stay given by the interval $\mathcal{T}^{(m)} = \{t_1^{(m)} + 1, t^{(m)} + t_1^{(m)}\}$, a pattern will be constructed for the representation of the multiple track scenes with varying track lengths. Clearly, $t_0 \leq t_1^{(m)} + 1 \leq t^{(m)} + t_1^{(m)} \leq t$, for the observation interval from t_0 to present time t. Define $\vec{x}^{(m)}(k)$ to be the parameter vector associated with the m^{th} target at time k given by $\{\vec{p}^{(m)}(k)\ , \vec{\phi}^{(m)}(k)\ , a^{(m)}(k)\} \in \left(\mathcal{M}(3) \times \Re^3 \times \mathcal{A}\right)$. The parameter set associated with the complete m^{th} track given $\mathcal{T}^{(m)}$ is

$$\{\vec{x}^{(m)}(k) : k \in \mathcal{T}^{(m)}\} \in \left(\mathcal{M}(3) \times \Re^3 \times \mathcal{A}\right)^{\mathcal{T}^{(m)}} \times \aleph \ .$$

The parameter vector for an M-track scene becomes the collection of each of the single track configurations, element of $\mathcal{X}_t(M)$ according to

$$\begin{aligned} \vec{x}_t(M) &= \bigcup_{m=1}^{M} \{\vec{x}^{(m)}(k) : k \in \mathcal{T}^{(m)}\} \\ \in \mathcal{X}_t(M) &\equiv \prod_{m=1}^{M} \left(\bigcup_{t^{(m)}=1}^{t} \left(\mathcal{M}(3) \times \Re^3 \times \mathcal{A}\right)^{t^{(m)}} \times \aleph \right) . \end{aligned}$$

Since M is unknown a-priori the complete parameter space is defined as $\mathcal{X}_t = \bigcup_{M=0}^{\infty} \mathcal{X}_t(M)$.

3 Bayesian Posterior

We take a Bayesian approach for solving the estimation problem by defining a posterior probability, which is the product of the prior density with the data likelihood, on the parameter space.

3.1 Prior

The prior measure encodes ***a-priori*** information about the parameters to be estimated. In particular this knowledge can come from, say, the airplane dynamics, or some previous knowledge of target type and number of targets. We utilize a set of Newton's second law based equations governing airplane motion to generate a prior density on the airplane positions as described in [11, 10]. This prior density is parameterized by the sequence of airplane orientations demonstrating the fundamental connections between the tracking and recognition algorithms.

In this work we utilize a simple Markov von-Mises prior on the orientation angles $\vec{\phi} = [\phi_1, \phi_2, \phi_3] \in \mathcal{M}(3)$.

3.2 Likelihood

There are two sensor types in this problem, a ***tracking sensor***, consisting of an array of passive sensors and a range radar, and a ***high-resolution imaging*** sensor.

Tracking

For the azimuth-elevation coordinate tracking a cross array of isotropic sensors is assumed [12, 10] using the standard narrowband signal model developed in [13]. Accordingly, the expression for sensor response to M signal sources at locations $\vec{p}^{(1)}(k), .., \vec{p}^{(M)}(k)$ with amplitudes $\vec{s}(k) = [s^{(1)}(k)\ , .., s^{(M)}(k)]$ is

$$\vec{y}_1(k) = \sum_{m=1}^{M} d(\vec{p}^{(m)}(k)) 1_{\mathcal{T}^{(m)}}(k) s^{(m)}(k) + \vec{n}_1(k)\ , \quad (1)$$

where $\vec{n}_1(k)$ is the 0-mean complex Gaussian noise vector of the Goodman class with covariance $\sigma_1^2 I$, $1_{\mathcal{T}^{(m)}}(k)$ selects the targets present in the scene at time k and $\vec{d}(\vec{p}^{(m)}(k))$ is the regular vandermonde direction vector corresponding to the m^{th} target. We focus on the estimation of the target positions by assuming the signal amplitudes to be known. The set of tracking data collected up to time t is given by $I_t^{\mathcal{P}}(1) \equiv \{\vec{y}_1(k) : k \in \{1, .., t\}\}$ and the likelihood of this data has the standard Gibbs potential

$$\begin{aligned} L_t^1(\vec{x}_t(M)) &= -\frac{1}{\sigma_1^2} \sum_{k=1}^{t} |\vec{y}_1(k) \\ &\quad - \sum_{m=1}^{M} d(\vec{p}^{(m)}(k)) 1_{\mathcal{T}^{(m)}}(k) s^{(m)}(k)|^2 \ . \end{aligned}$$

where $|\cdot|$ is the vector 2-norm.

Imaging

While the statistical models for high-resolution radar imaging are being incorporated in this problem by others ([14, 15, 16]), all of the results shown here are based on an optical imaging system. In this system, data is a sequence of 2-D images resulting from projecting the 3-D volume containing the targets onto the focal plane of optical imaging sensor. Since the parameter set $\vec{x}_t(M)$ completely parameterizes the imaged volume we can write the projection as a deterministic operation from the parameter space to a discrete 2-D lattice $\Re^{\mathcal{L}}$, i.e.

$$\mathcal{P} : \mathcal{X}_t \rightarrow \Re^{\mathcal{L}}_{+}.$$

For the implementation presented here a white Gaussian noise model was used, with the measured data $\vec{y}_2(k)$ for the set of M targets having mean given by the projection of M targets $\mathcal{P}(\vec{x}_t(M))$. The likelihood potential is given by

$$L_t^2(\vec{x}_t(M)) = -\frac{1}{2\sigma_2^2}\sum_{k=1}^{t} ||\vec{y}_2(k) - \mathcal{P}(\vec{x}_t(M))||^2 \quad (2)$$

where $||\cdot||$ represents matrix 2-norm and σ_2^2 is the noise variance.

The posterior density in Gibbs form can be written as

$$\begin{aligned} \pi_t(\vec{x}_t(M)) &= \frac{1}{Z}e^{-E_t(\vec{x}_t(M))} \\ &= \frac{1}{Z}e^{-(P_t(\vec{x}_t(M))+L_t(\vec{x}_t(M)))} \quad , \end{aligned}$$

where $L_t(\vec{x}_t(M)) = L_t^1(\vec{x}_t(M)) + L_t^2(\vec{x}_t(M))$ is the potential associated with data likelihood and $P_t(\vec{x}_t(M))$ is the potential associated with the prior density on the parameter space $\mathcal{X}_t$.

4 Estimation by Random Sampling

Our approach is to construct a jump-diffusion Markov process, following the analysis outlined in [8], having the limiting property that it converges in distribution to the Bayes posterior. Following the *jump-diffusion* dynamics the process (i) on random exponential times jumps from one of the countably infinite set of subspaces to another estimating discrete parameters, and (ii) between jumps it performs diffusion following the S.D.E.'s appropriate for the current subspace. In this sequential estimation problem the posterior density changes at each observation time due to the addition of one more data sample to the data set. Therefore, at any given time t the sampling process generates samples from the posterior density having the Gibbs energy $E_t(\vec{x}_t(M))$. This is proven by showing that the backward kolmogoroff operator associated with the Markov process satisfies the condition of stationarity given by $\int_{\mathcal{X}_t} Af(\vec{x}_t(M))\pi(d\vec{x}_t(M))d\vec{x}_t(M) = 0$ as verified in [17]. The jump-diffusion Markov process $\{X(s), s \geq 0\}$ samples from the posterior density $\pi_t(\vec{x}_t(M))$ as follows.

4.1 Jump Process

The jump process contributes in the search of discrete parameters like number of the targets, length of the tracks, and target types. It deals with the assignment of tracks and the choice of model order in two ways. First, the individual tracks are developed by probabilistically placing the track-segments sequentially in the associated track configuration. Secondly, the jump process moves among the subspaces of variable numbers of tracks via the addition and deletion of tracks. Our implementation is based on the jump moves derived from a modification of the Metropolis algorithm introduced in 1953 [18]. This algorithm is implemented through following steps:

Metropolis Based Jump-Algorithm

```
1. Generate independent exponential r.v.'s $u_1, u_2, ..$ with the intensity $\lambda$, where $\lambda$ is the average number of diffusion cycles for every jump move.
2. At time $t_i = \sum_{j=1}^{i} u_j$ draw a candidate $\vec{y}_t(M')$ from the prior (e.g by using the equations of motion).
3. Compute $L_t(\vec{y}_t(M'))$.
     If $[L_t(\vec{x}_t(M)) - L_t(\vec{y}_t(M'))] > 0$,
         go to $\vec{y}_t(M')$.
     else
         go to $\vec{y}_t(M')$ with the probability $e^{-[L_t(\vec{y}_t(M'))-L_t(\vec{x}_t(M))]}$.
4. Repeat step 2.
```

The jump parameters corresponding to this algorithm satisfy the condition of stationarity as proven in [17].

4.2 Diffusion Process

The diffusion process contributes in the search of the features which lie in the continuous space, positions $\vec{p}(k)$ and orientations $\vec{\phi}(k)$. It is a sample path continuous process which essentially performs a randomized gradient descent on the posterior potential $E_t(\vec{x}_t(M))$ in the current subspace $\mathcal{X}_t(M)$ according to Langevin's stochastic differential equation (SDE). This process has two parts $X(s) = [X_1(s)\ X_2(s)]$ described as follows. Define $n_M = \sum_{m=1}^{M} t^{(m)}$, the total track segments in $\vec{x}_t(M)$ and associate with the first $3n_M$ components of the S.D.E. $X(s)$ the flow through $\mathcal{M}(3)^{n_M}$ to estimate the orientations, and the last $3n_M$ components the flow through $\Re^{(3n_M)}$ to estimate the positions. Then the diffusion $X(s)$ satisfies the following vector S.D.E.:

$$X_1(s) = [X_1(0) + \int_0^s -\frac{1}{2}\nabla_1 E_t(X(\tau))d\tau + W_1(s)]_{\mathrm{mod}\ 2\pi}, \tag{3}$$

$$X_2(s) = X_2(0) + \int_0^s -\frac{1}{2}\nabla_2 E_t(X(\tau))d\tau + W_2(s), \tag{4}$$

where $[\cdot]_{\mathrm{mod}\ 2\pi}$ is taken componentwise, $W_1(s), W_2(s)$ are standard vector Wiener processes of the dimensions $\mathcal{M}(3)^{n_M}, \Re^{3n_M}$ respectively, and ∇_1, ∇_2 are the gradients with respect to the position and orientation parameters.

Now we utilize a result on the ergodic properties of jump-diffusion process from [17]. This result verifies that the jump-diffusion process constructed above samples from the posterior distribution $\pi_t(\vec{x}_t(M))$. As a consequence the sample average $(\frac{1}{N}\sum_{n=1}^{N} X(n\Delta))$ of the Markov process converges to the conditional mean as $N \rightarrow \infty$, which is the *MMSE* estimate.

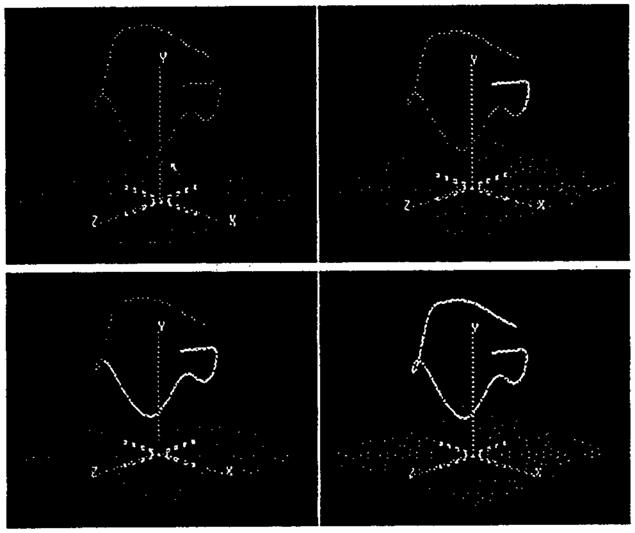


Figure 1: 3D track estimation: The left panel shows the actual track drawn in gray with the mesh representing ground supporting the observation system in the inertial frame of reference. The other panels display the results from various stages of the single track estimation with the estimates drawn in white.

5 Implementation Results

The algorithm for estimating the motion of a single target (M = 1) was jointly implemented using a Silicon Graphics workstation for data generation and visualization, and a massively parallel 4096 processor SIMD DECmpp machine for implementing the track-recognition algorithm. The *flight simulator* software on Silicon Graphics workstation was utilized to generate parameterized airplane paths. The algorithm proceeds via a sequence of jump moves corresponding to the births of track-segments and adjusting the track-estimates between the jumps via diffusion algorithm. Shown in Fig 1 is the result for the complete track estimation algorithm with the top-left panel showing the simulation environment of the implementation. The mesh represents the ground supporting the inertial frame of reference and the sensor systems while grey track is the parameterized plane path generated from the flight simulator on Silicon Graphics and used as true track in the simulations. The other three panels show algorithm at various stages of estimation with the estimates drawn overlapping in white. The final result is shown in the bottom right panel.

Acknowledgment

This research was supported by ONR N00014-92-J-1418, ARO DAA 03-92-6-0141, and Rome Laboratory F30602-92-C-004.

References

[1] Y. Bar-Shalom and E. Tse. Tracking in cluttered environment with probabilistic data association. *Automatica*, (11):451–460, 1975.

[2] Y. Bar-Shalom and T. E. Fortmann. *Tracking and Data Association*. Academic Press, 1988.

[3] Editor: Y. Bar-Shalom. *Multitarget-Multisensor Tracking*. Artech House, 1990.

[4] C. R. Rao, C. R. Sastry, and B. Zhou. Tracking the direction of arrival of multiple moving targets. *IEEE Transactions on Acoustics,Speech and Signal Processing*, accepted for publication.

[5] C. K. Sword, M. Simaan, and E. W. Kamen. Multiple target angle tracking using using sensor array outputs. *IEEE Trans. AES.*, 26(2):367–373, 1990.

[6] A. L. Swindlehurst and T. Kailath. Passive direction-of-arrival and range estimation for near-field sources. In *Proc. IEEE ICASSP*, pages 123–128, 1988.

[7] Y. Amit, U. Grenander, and M.I. Miller. Ergodic properties of jump-diffusion processes. *Annals of Applied Probability*, submitted December 1992.

[8] U. Grenander and M. I. Miller. Representations of knowledge in complex systems. *Journal of the Royal Statistical Society*, 56(3), 1994.

[9] U. Grenander. Advances in pattern theory: The 1985 rietz lecture. *The Annals of Statistics*, 17:1–30, 1985.

[10] M.I. Miller, R. S. Teichman, A. Srivastava, J.A. O'Sullivan, and D. L. Snyder. Jump-diffusion processes for automated tracking-target recognition. In *1993 Conference on Information Sciences and Systems*, Baltimore, Maryland, March 24-26 1993. Johns Hopkins University.

[11] A. Srivastava, N. Cutaia, M. I. Miller, J. A. O'Sullivan, and D. L. Snyder. Multi-target narrowband direction finding and tracking using motion dynamics. In *Proc. 30th Annual Allerton Conference on Communication, Control, and Computing*, pages 279–288, Urbana, Il, October 1992. University of Illinois.

[12] M.I. Miller and D. R. Fuhrmann. Maximum likelihood narrow-band direction finding and the em algorithm. *IEEE Acoust. Speech and Signal Processing*, 38, No.9(38, No.9):560–577, 1990.

[13] R. Schmidt. *A signal subspace approach to multiple emitter location and spectral estimation.* Ph.D. Dissertation of Stanford University, Palo Alto, CA., Nov. 1981.

[14] D.L. Snyder, J.A. O'Sullivan, and M.I. Miller. The use of maximum-likelihood estimation for forming images of diffuse radar-targets from delay-doppler data. *IEEE Transactions on Information Theory*, pages 536–548, 1989.

[15] J.A. O'Sullivan, P. Moulin, and D.L. Snyder. Cramer-rao bounds for constrained spectrum estimation with application to a problem in radar imaging. In *Proceedings 26th Allerton Conference on Communication, Control, and Computing*, Champaigne, Urbana, October 1988 October 1988. Urbana, IL.

[16] M.I. Miller, D.R. Fuhrmann, J.A. O'Sullivan, and D.L. Snyder. Maximum-likelihood methods for toeplitz covariance estimation and radar imaging. In Simon Haykin, editor, *Advances in Spectrum Estimation.* Prentice-Hall, 1990.

[17] A. Srivastava. *Automated Target Tracking and Recognition Using Jump Diffusion Processes.* M. S. Thesis, Washington University,, St. Louis, Missouri, October 1993.

[18] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Physical Chemistry*, 21:1087, 1953.

A Likelihood-Based Approach to Joint Target Tracking and Identification

J. A. O'Sullivan S. P. Jacobs M. I. Miller D. L. Snyder
Electonic Systems and Signals Research Laboratory
Department of Electrical Engineering
Washington University
St. Louis, MO 63130

Abstract

The identification of aircraft using high range resolution radars requires a close interaction between the tracker and the identification algorithm. The tracker produces as output not only the estimates of target position and orientation, but also a measure of the quality of those estimates. The identification algorithm performs a likelihood-based search over both target types and target positions and orientations. This search compares the measured data to predicted data obtained from surface models of the possible targets. The search constitutes a simultaneous target tracking and identification system.

1 Introduction

Previous papers have presented a general approach to likelihood-based tracking and recognition [1, 2, 3]. The fundamental viewpoint is that the incorporation of tracking data into the recognition algorithm improves recognition performance and incorporating target aspect angle information from the recognition algorithm improves tracking. The algorithm is likelihood based implying that the optimum estimate of target type is the most likely given both the tracking and recognition (high range resolution radar) data; the optimum track is determined from a random sampling of the posterior on the sequence of states given the measurements.

This paper examines in more detail the domain over which range profiles are defined and the impact of this on the recognition algorithm. In particular, range profiles are viewed as being a mapping from a pair of angles to time functions. These two angles live on a sphere with the north and south poles identified. The topology of this space impacts the recognition algorithm and the set of unambiguous target orientations that can be discriminated based solely on range profiles.

We have proposed using diffusion and jump-diffusion searches for target tracking and recognition [3, 4, 5]. While results have appeared discussing the implementation of diffusions on manifolds such as spheres, those results tend to be less accessible than diffusions on lattices. This motivates a discussion of a diffusion over a lattice of points on the set of range profiles. Convergence of the algorithm in measure is shown based on arguments from Markov chains.

2 Data Likelihoods

In [1, 2] a model for range profiles is introduced. This model may be viewed as mapping the orientation of a target to time functions. As discussed in the next section, this mapping depends on two angles (θ', ϕ') which can be determined from the complete orientation. In general, a sequence of range profiles are measured. Denote the kth range profile by $b(t; \theta'_k, \psi'_k)$. The available data are

$$r_k(t) = \int h(t-\tau) b(\tau; \theta'_k, \psi'_k) d\tau + w_k(t) \qquad (1)$$

where $h(t)$ is determined by the complex envelope of the transmitted signal and $w_k(t)$ is complex white Gaussian noise with intensity N_0. For simplicity, rewrite $r_k(t)$ as

$$r_k(t) = s(t; \theta'_k, \psi'_k) + w_k(t) \qquad (2)$$

where s is the signal part of the received waveform. The likelihood function for $r_k(t)$ is

$$
\begin{aligned}
L(r_k|\theta'_k,\psi'_k) &= exp\Big(\frac{1}{N_0}Re\Big[\int[r_k(t) \qquad (3)\\
&\quad -\frac{1}{2}s(t;\theta'_k,\psi'_k)][s(t;\theta'_k,\psi'_k)]^* dt\Big]\Big).
\end{aligned}
$$

The tracking measurements take place on a longer time scale than the range profiles are collected. In the S-Band radar at Rome Laboratory, one tracking measurement is made for each range profile. The tracking data consist of estimated 3-D positions of the target. These estimates typically ignore target dynamics and may be thought of as equal to the actual target positions plus additive noise whose variance is determined by such factors as the width of the autocorrelation function of the transmitted tracking pulse, the curvature of the antenna pattern, the tracking modality, and the actual receiver noise. To simplify the analysis, assume one tracking measurement for each range profile measurement, and let y_k in R^3 be the kth data vector,

$$y_k = x_k + n_k; \qquad (4)$$

here, n_k is a sequence of i.i.d. Gaussian random vectors and x_k is the actual postition at time k.

In typical state space models for targets, the state is 12 dimensional, consisting of the positions, velocities, orientations, and rotation rates [7]. These equations are often linearized and reduced in dimension in order to make the model more amenable to standard (linear) state estimation procedures. In any event, the state space model defines a likelihood (prior) for the states at the instants of time at which the measurements take place. The vector x_k constitutes three of the states, and the angles θ'_k and ψ'_k determine two orientation angles (and are clearly determined by the three orientations in the state vector). The remaining states are not measured directly. Let ξ_k denote the full state vector at time k and $p(\xi_k|\xi_{k-1})$ denote the conditional density function for the states at time k given the states at time $k-1$. Then the posterior on the states given the measurements for $k = 0, 1, ..., K$ is proportional to

$$p(\xi_0)\prod_{k=1}^{K} p(\xi_k|\xi_{k-1})L(r_k|\theta'_k,\psi'_k)p(y_k|x_k). \qquad (5)$$

The estimates for these states are then obtained from this likelihood function. The algorithms proposed by the authors implement diffusions on the state space to randomly sample from the posterior.

The discussion above assumes that the target is known. If the target is unknown, the algorithm first conditions on target type, estimates the states for that target type, then jumps to different target types. The likelihoods for each target type are stored, and after the algorithm has run sufficiently long, the estimate of target type is declared to be that type that maximizes likelihood. For more on the jump-diffusion algorithm, see [3, 4]; the continuous diffusions for state estimation are discussed in [5].

3 Orientation Ambiguity

Suppose we are given the problem of estimating the orientation of a known object from a single observation of the range profile. Assume that $s(t;\theta',\psi')$ is known for all (θ',ψ'). In this section, we justify the dependence of s on only two angles and discuss the types of errors in orientation estimation that arise. Previous work by Sworder, et.al. [6] identifies both local and global errors arising out of the problem of target identification through matching a noisy observation with a stored replica, when the data to be compared are 2-D silhouettes of the target. For range profile data, the problem is analogous. Two types of global errors arise from symmetry and ambiguity. The global errors are deterministic and correspond to the fact that different orientations of the target yield equal range profiles under our model. Local errors arise due to limits in estimation accuracy as determined, for example, by the Cramer-Rao lower bound.

Since the target recognition and tracking algorithms share data, we define the geometry of our experiment consistent with the treatment of rigid body dynamics used in the tracker (see Friedland [7]). The target can be described by a set of points in one of two spaces: an inertial frame of reference which is assumed fixed, and a body-centered frame of reference that is independent of orientation. Each frame of reference is described by three orthogonal axes with respect to a common origin at the center of the target. The inertial x-axis is assumed to be the radar line-of-sight; the body-centered x, y, and z axes point to the front, right side, and bottom of the target respectively.

The orientation of a target is defined by the orientation of its body axes with respect to the inertial axes. An arbitrary orientation can be described by a set of three angles (ϕ,θ,ψ), called roll, pitch, and yaw angles. These angles define an ordered triple of rotations from the inertial to the body-centered frame of reference as follows:

$$\begin{bmatrix} x_B \\ y_B \\ z_B \end{bmatrix} = T_{IB} \begin{bmatrix} x_I \\ y_I \\ z_I \end{bmatrix} \tag{6}$$

$$T_{IB} = R_x(\phi)R_y(\theta)R_z(\psi) \tag{7}$$

where

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & sin(\phi) \\ 0 & -sin(\phi) & cos(\phi) \end{bmatrix} \tag{8}$$

and $R_y(\theta)$ and $R_z(\psi)$ are defined analogously (see Friedland [7]). The ordering of the rotations is implied by the above matrix product: yaw then roll then pitch.

Many targets of interest such as aircraft exhibit symmetry with respect to the body-centered x-z plane. Assume that the reflectivity of the target is given by the summed contribution of a finite number of scattering sites, and the range profile results from projection of the individual reflectivities along the line of sight and sorting into range bins. Under these conditions the range profile will be invariant to reflection in the inertial x-z plane yielding the potential global symmetry error. Even without the symmetry, there is global ambiguity. This is because, for a target in an arbitrary orientation, the range profile is invariant to rotation about the inertial x-axis. Global errors are unresolvabe without including the prior arising from the state space description of the target kinematics.

Define an equivalence class to be the set of orientations of a target that produce identical range profiles. Given an orientation for a target, other orientations in the same equivalence class arise by rotating about the x_I axis. Since x_I is not in general aligned with any of the body axes, a three step process is employed.

1. Perform a series of rotations about major axes so that the x_I axis is aligned with one of the body axes. There are many possible solutions, but we will choose to simply invert the original roations T_{IB}.

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = T_{IB}^{-1} \begin{bmatrix} x_B \\ y_B \\ z_B \end{bmatrix} = \begin{bmatrix} x_I \\ y_I \\ z_I \end{bmatrix} \tag{9}$$

2. Perform and arbitrary rotation about the x_I axis. Let the angle of rotation be α. Since the x_I and x_B axes are now aligned, the necessary rotation is simply $R_x(\alpha)$.

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = R_x(\alpha) \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = R_x(\alpha) \begin{bmatrix} x_I \\ y_I \\ z_I \end{bmatrix} \tag{10}$$

3. Invert the rotations applied in step 1.

$$\begin{bmatrix} x_3 \\ y_3 \\ z_3 \end{bmatrix} = T_{IB} \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} \tag{11}$$

The ordering of rotations presented above makes good physical sense. However, it is possible to achieve all orientations regardless of this ordering. For example, suppose we chose to perform roll first.

$$\begin{bmatrix} x_B \\ y_B \\ z_B \end{bmatrix} = R_y(\theta)R_z(\psi)R_x(\phi) \begin{bmatrix} x_I \\ y_I \\ z_I \end{bmatrix} \tag{12}$$

This choice of ordering makes it easy to determine all orientations in an equivalence class. If we now apply our three step process:

$$\begin{bmatrix} x_3 \\ y_3 \\ z_3 \end{bmatrix} = R_y(\theta)R_z(\psi)R_x(\phi)R_x(\alpha) \begin{bmatrix} x_I \\ y_I \\ z_I \end{bmatrix} \tag{13}$$

Since $R_x(\phi)R_x(\alpha) = R_x(\phi + \alpha)$, this equation has the interesting implication that if we set $\alpha = -\phi$, the associated equivalence class for each orientation has at least one member with $\phi = 0$. Thus, every equivalence class is uniquely identified by a pair of angles θ' and ψ' which are the pitch and yaw of that member of the equivalence class with zero roll. The rotations, in standard order, in equation (11) define an equivalence class. By our previous argument there must be some angle α and some pair (θ', ψ') such that the following equation is satisfied:

$$R_y(\theta')R_z(\psi')R_x(-\alpha) = R_x(\phi)R_y(\theta)R_z(\psi) \tag{14}$$

The three angles that solve this matrix equation must satisfy each of the nine scalar equations contained within. However, if we are interested only in the angles (θ', ψ') that identify the equivalence class for a given orientation, we may consider the three scalar equations from the first column, which are independent of α.

$$cos(\theta')cos(\psi') = cos(\theta)cos(\psi) \tag{15}$$

$$-\,sin(\psi') = -cos(\phi)sin(\psi) + sin(\phi)sin(\theta)cos(\psi) \tag{16}$$

$$sin(\theta')cos(\psi') = sin(\phi)sin(\psi) + cos(\phi)sin(\theta)cos(\psi) \tag{17}$$

The unique solution to these equations defines a mapping from a complete orientation to a pair of angles that identify an equivalence class:

$$\theta' = arctan\left(\frac{sin(\phi)sin(\psi) + cos(\phi)sin(\theta)cos(\psi)}{cos(\theta)cos(\psi)}\right) \tag{18}$$

$$\psi' = arcsin\left(cos(\phi)sin(\psi) - sin(\phi)sin(\theta)cos(\psi)\right). \tag{19}$$

All orientations that can be achieved through yaw and pitch only have coordinates (θ', ψ') that live in the space $[-\pi, \pi]^2$. However, for every point in this space there is at least one other point that has the same range profile. For example, the orientations $(0, 0)$ and (π, π) will clearly be in the same equivalence class. Further investigation shows that all orientations for which $\psi' = \pm\frac{\pi}{2}$ are likewise equivalent. If one considers all points in $\{(\theta', \psi') \in [-\pi, \pi]^2\}$ and identifies together all sets of points that lie in the same equivalence class, a topology is suggested like that of a sphere, except that the poles of this sphere are identified together. One can imagine deforming a sphere by drawing the poles together to meet at the origin (see figure 1). Such an object resembles a torus with inner radius of zero, but the singularity at $\psi' = \pm\frac{\pi}{2}$ makes the topology of this space much more like that of a sphere.

4 Sampling of the Likelihood Surface

For the discussion in this section, consider the problem of finding the maximum likelihood estimate of the equivalence class of orientations of the target from a single range profile measurement. This implies finding the maximum of $L(r_k|\theta'_k, \psi'_k)$. Assume that the known range profiles $s(t; \theta'_k, \psi'_k)$ are in fact a stored set of observations or precomputed simulations on a grid of points for (θ'_k, ψ'_k). The question is how to choose the grid. Two opposing considerations constrain the number of range profiles to be made available. The high variablility of the range profile with respect to orientation indicates a need for a fine sampling of the space of equivalence classes. Storage and computational considerations restrict the number of range profiles that can be searched over in a reasonable time.

Since the space of equivalence classes has a topolgy similar to that of a sphere, we consider the problem of choosing a discrete lattice on this space in terms of choosing points on the surface of a sphere. We would like this set of points to possess certain properties. Clearly one of these is uniformity; we would like the set of points to be equidistant from each other in terms of a distance measure on the sphere. A set of points that form the vertices of a regular polyhedron inscribed in the unit sphere exhibit uniformity in that adjacent pairs of vertices are equidistant in cord length, measured along the edge of the polyhedron that connects them, and in arc length. Additionally, each face of a regular polyhedron cuts an equal solid angle, or a patch of equal area in the surface of the circumscribing sphere. If a polyhedron with triangular faces such as the octahedron or icosahedron is chosen, then every vertex has a well defined set of nearest neighbors, and is connected to all nearest neighbors along edges. This turns out to be a useful property when searching through a set of points on the sphere.

The clear problem with the above strategy is that there are a very limited number of regular polyhedra. However, if we relax our constraint to require only approximate uniformity, then a wide variety of polyhedra are available whose vertices represent the sampling of a sphere. Geodesic architecture has employed such methods for decades in designing structures that provide shelter without central support. Kenner [8] provides a simple algorithm for computing the (θ', ψ') of any point on a geodesic polyhedron from three integers that index the location on the regular polyhedron from which they are derived. Maximum uniforimtiy of the sampling is obtained by subdividing the regular polyhedron whose faces are closest to the sphere, namely the icosahedron. Figure 1 shows a geodesic decomposition of the icosahedron to which a deformation has been applied, so that the poles are identified.

5 Stochastic Search on the Sphere

Having defined a grid of points on the sphere, the problem of estimating the orientation equivalence class reduces to determining the grid point with highest likelihood. Motivated by diffusion searches as discussed in [3,4,5], we define a discrete-time stochastic search on the grid points. At time n in the iteration, compute the likelihood $L(r_k|\theta'_l, \psi'_l)$ for (θ'_l, ψ'_l) in a neighborhood of the present guess (θ'_n, ψ'_n). Move to

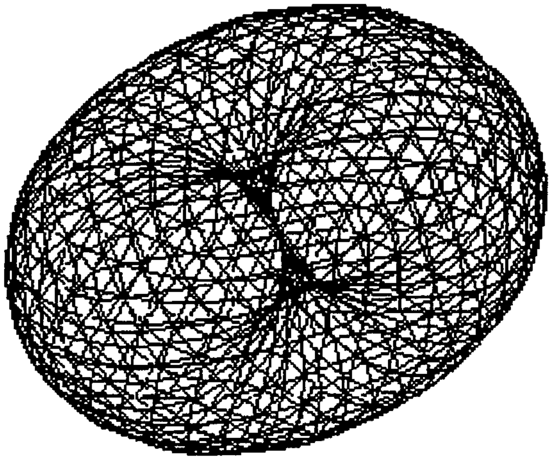


Figure 1: Topology of the space of equivalence classes. This object is a polyhedron inscribed in a sphere, which is then deformed so that the poles are identified.

(θ'_l, ψ'_l) with probability

$$p_{n,l} = \frac{L(r_k|\theta'_l, \psi'_l)}{\sum_{m \in N(n)} L(r_k|\theta'_m, \psi'_m)} \tag{20}$$

where $N(n)$ is the neighborhood grid points of n. For example, the neighborhood could consist of all grid points closest to the present grid point. This algorithm defines a Markov chain with state corresponding to the present grid point. If the neighborhoods are completely connected (for some M any grid point may result from the above algorithm in M steps starting from any other grid point, with positive probability), then the Markov chain is recurrent. Using the usual argument based on the Perron-Frobenius Theorem, there is a unique limit measure of this algorithm. This measure is given by the positive left eigenvector of the transition matrix with entries $p_{n,l}$. This measure does not necessarily equal the posterior. As the number of grid points increases, however, the algorithm becomes more accurate.

6 Summary and Conclusions

Within the context of joint tracking and recognition of aircraft, this paper has examined in more detail the issues associated with using range profiles for the recognition. In particular, to accomplish the recognition it is tacitly assumed that improved tracking based on better estimates of the orientations over time is required. A set of equivalence classes of orientations is derived. If the search over orientation is to be performed on a discrete grid, a discretization based on geodesics is explored. Finally, a stochastic search over the grid that converges in measure is proposed.

Acknowledgment

This work was supported by Rome Laboratory under contract F30602-92-C-0004.

References

[1] J. A. O'Sullivan, K. C. Du, R. S. Teichman, M. I. Miller, D. L. Snyder, and V. C. Vannicola, "Reflectivity Models for Radar Target Recognition," *SPIE OE/Aerospace Science and Sensing Conf. Auto. Object Recog. III*, Orlando, FL, April 1993.

[2] J. A. O'Sullivan, K. C. Du, R. S. Teichman, M. I. Miller, D. L. Snyder, and V. C. Vannicola, "Radar Target Recognition Using Shape Models," *Proc. 30th Allerton Conf. Comm., Control, and Computing*, Urbana, IL, pp. 515-523, October 1992.

[3] A. Srivastava, N. Cutaia, M. I. Miller, J. A. O'Sullivan, and D. L. Snyder, "Multi-Target Narrowband Direction Finding and Tracking Based on Motion Dynamics," *Proc. 30th Allerton Conf. on Communication, Control, and Computing*, Urbana, IL, pp. 279-288, October 1992.

[4] M. I. Miller, R. Teichman, A. Srivastava, J. A. O'Sullivan, and D. L. Snyder, "Jump-Diffusion Processes for Automated Tracking-Target Recognition," *Proc. 27th Conf. Info. Sci. and Sys.*, Johns Hopkins University, pp. 617-622, March 1993.

[5] J. A. O'Sullivan, M. I. Miller, A. Srivastava, and D. L. Snyder, "Tracking Using a Random Sampling Algorithm," *Proc. 12th World Congress of the International Federation of Automatic Control*, vol. 5, Sydney, Australia, pp. 435-438, July 1993.

[6] D. D. Sworder and P.F. Singer and D. Doria and R. G. Hutchins "Image-Enhanced Estimation Methods" *Proc. IEEE*, Vol. 81, pp. 796-811, 1993.

[7] B. Friedland *Control System Design*. New York: McGraw-Hill, 1986.

[8] H. Kenner *Geodesic Math and How to Use it*. Los Angeles: University of California Press, 1976.

Reflectivity Models for Radar Target Recognition

Joseph A. O'Sullivan[1]
K. Cecilia Du[1]
Robert S. Teichman[1]
Michael I. Miller[1]
Donald L. Snyder[1]
Vincent C. Vannicola[2]

[1]Electronic Systems and Signals Research Laboratory
Department of Electrical Engineering
Washington University
St. Louis, MO 63130

[2]Rome Laboratory
Griffiss AFB, NY 13441-5700

ABSTRACT

A model for the generation of reflectivity profiles is presented for use in a radar target recognition system. The data are assumed to come from two sensors: a high range resolution radar and a tracking radar. The object is simultaneously tracked and identified using estimation theoretic methods by comparing a sequence of received range profiles to range profiles generated from surface templates. The tracking data are used to form priors on the position and orientation of the object. The templates consist of surface descriptions comprised of electromagnetically large patches tiling the entire object. The predicted return is computed from several quantities. First, the reflectivity range profile is computed from the patches incorporating a shading function. The physical optics approximation is that patches not directly illuminated by the transmitted signal do not contribute to the return signal. Second, the reflected signal is approximated by the convolution of the transmitted signal with the range profile. Third, the receiver design yields the actual I-Q data available for processing.

The tracking is performed using advanced models of aircraft of interest. Given measurements of the bulk position and velocity of the aircraft, the other states (orientation, angular rates, other velocities) are estimated. These estimates and a quantitative measure of the quality of the estimates (as provided, for example, by an extended Kalman filter) are used in the prediction of the range profiles. They also may be used in a preliminary object discrimination step based solely on the dynamics.

The models for the available data are based on radars in use at Rome Laboratory in Rome, NY. The high range resolution data comes from an S-band radar which transmits chirp pulses and uses stretch processing in the receiver. The processing is performed on a DEC/MPP computer with 4096 processors. Preliminary results are encouraging.

This work was supported by Rome Laboratory under contract number F30602-92-C-0004.

1. INTRODUCTION

This paper presents recent research in the automatic recognition of targets using sophisticated shape models. The target is assumed to be tracked by one radar sensor and to have high resolution radar data collected by a secondary sensor. The high resolution data are used to identify the target as well as improve tracking performance. In order to improve the tracking, an explicit model for the relative sensor positions (and their resulting measurements) is included. A quantitative estimate for the improvement in tracking performance results. The results are based on standard radar literature [1,2,3,4,5,6,7,8,9].

The paper starts with a review of some results from [10]. In that paper, a rather detailed description of simulated range profiles of targets is presented. Range profiles computed using that algorithm are included here. The intention is to use these simulated range profiles in a target recognition algorithm. The algorithm as presented here takes advantage of the tracking data to assist in the target recognition. A detailed likelihood model for the available tracking data is presented. This likelihood is used both for tracking and for recognition.

2. RADAR RANGE PROFILES

The prediction of range profiles from surface templates has been proposed in [10]. The model follows terminology and analysis from [6,7,8]. The computations may be viewed as being based on shape models [11]. Here, the template is a surface model of the object consisting of patches that tile the surface. Each patch has a three dimensional location vector, a three dimensional normal vector, an area, and a type label. The reflectivity of a patch is computed conditioned on the position and orientation of the target. The patch is rotated by rotating the location and normal vectors. The patch is translated by translating the location vector.

For patch k, assume that ξ_k is the angle between the direction of propagation and the normal vector. Let A_k be the area of the patch. The reflectivity from a patch is computed using a physical optics approximation for the radar cross section of the patch. Initially, we assume the patch is a circular disk of area $\pi r_k^2 = A_k$. The resulting reflectivity is proportional to

$$f(\xi_k, A_k) = \frac{A_k}{2\pi r_k} \tan \xi_k J_1\left(\frac{4\pi r_k}{\lambda} \sin \xi_k\right), \tag{1}$$

where J_1 is a Bessel function, and λ is the wavelength of the transmitted signal. Later, more accurate models may be incorporated. (Note that this is slightly different than the presentation in [10]. This approximation reduces at normal incidence to

$$f(0, A_k) = \frac{A_k}{\lambda} .$$

The RCS at normal incidence is $4\pi A_k^2/\lambda^2$, which is 4π times f^2, and is consistent with f being the reflectivity.)

Each patch has an associated shading gain g_k. Here, we let $g_k = 0$ if the patch is shaded, and $g_k = 1$ if it is not shaded. Assume that the patch, after rotation and translation, has x-coordinate $x_{I,k}$ (where I denotes inertial reference frame) and that the direction of propagation is along the x- axis. The reflectivity range profile of the target for the mth illumination is given by

$$b_m(\tau) = \sum_k g_{k,m} f(\xi_{k,m}, A_k) \exp[j(4\pi f_0 x_{I,k,m} + \theta_k)] \delta(\tau - 2x_{I,k,m}/c) , \tag{2}$$

where τ is the two-way delay to a patch at x on the target, $\delta(\cdot)$ is a Dirac delta function, θ_k is a (possibly random) phase associated with the kth patch, and additional subscripts "m" have been added to indicate the time dependence of certain quantities. Note that this range profile is conditioned on knowledge of the position and orientation of the patch during the mth illumination.

The range profiles $b_m(\tau)$ are not observed directly. Assume the complex envelope of the transmitted signal is $\tilde{s}_H(t)$, where the subscript H denotes the high resolution radar. If the target does not rotate significantly during an illumination, the complex envelope of the mth received signal equals the convolution of $\tilde{s}_H$ with b_m. Taking into account receiver noise,

$$\tilde{r}_m(t) = \int \tilde{s}_H(t-\tau)b_m(\tau)d\tau + \tilde{w}(t)\,, \tag{3}$$

where $\tilde{w}$ is complex white Gaussian noise with intensity N_0. The likelihood for r_m conditioned on the position and orientation is

$$L(\tilde{r}|\mathbf{x}) = \frac{1}{N_0}\mathrm{Re}\Big[\int(2\tilde{r}_m(t) - \int \tilde{s}_H(t-\tau)b_m(\tau)d\tau)(\int \tilde{s}_H(t-\tau)b_m(\tau)d\tau)^{*}dt\Big]. \tag{4}$$

A sufficient statistic is given by

$$\int \tilde{r}_m(t)\tilde{s}^{*}_H(t-\tau)dt\,. \tag{5}$$

Conventional receivers compute (5) (or something close to it).

Figures 1 and 2 show amplitudes of range profiles of an experimental aircraft computed using these methods. Figure 1 shows the aircraft and one range profile computed with the direction of propagation being parallel to the axis of the fuselage. The transmitted signal is a chirp, and stretch processing [3] is used in the receiver. For the simulations, the aircraft is assumed to be 10 meters long and the bandwidth of the chirp is 240 MHz. Figure 2 shows a sequence of range profiles computed as the target rotates through an angle of 20 degrees. The range profiles are shifted so that the centers of mass of all profiles are aligned.

3. TARGET RECOGNITION FROM RANGE PROFILES

There are two aspects of the target recognition algorithm: determining the target and estimating the positions and orientations of the target over time. As discussed in [11,12,13] an algorithm based on jump-diffusions has been proposed for solving this type of problem. The algorithm jumps between hypotheses of the target type. Then, conditioned on a specific target, a diffusion algorithm is used to estimate target states. After the algorithm has run for some period of time, a declaration of target type and estimates of the corresponding states may be obtained.

The state estimates are obtained from the joint likelihood for the high resolution data and the tracking data. This joint likelihood includes not only the individual effects of the sensors, but also their relationship. The next couple of sections explore the tracking radar likelihood in more detail.

4. MODEL FOR TRACKING RADAR

The tracking radar collects data from radar T. The target is modeled as a point target by the tracker at distance R or two-way propagation delay $\tau = 2R/c$ where c is the speed of propagation. The reflectivity of the target, $\tilde{\beta}$, is modeled as a complex Gaussian random variable with zero mean and variance σ^2, independent of distance to the target. In order to account for the decrease in amplitude of the reflectivity as a function of distance, a factor of $1/R^2$ is included in the model.

Part of the tracking error arises from the uncertainty in azimuth and elevation due to the antenna pattern, $A(\mathbf{u})$, which is expressed as a function of the direction vector

$$\mathbf{u} = [\sin\phi\cos\psi \;\; \sin\phi\sin\psi \;\; \cos\phi]^T \,, \tag{6}$$

where ϕ is elevation, measured with respect to the z-axis, and ψ is azimuth, measured in the $x-y$ plane. Note that the direction $\mathbf{u}$ corresponds to a rotation of the z-axis first in elevation, then in azimuth:

$$\mathbf{u} = R_\psi(\psi)R_\phi(\phi)\begin{bmatrix}0\\0\\1\end{bmatrix}, \tag{7}$$

where

$$R_\psi(\psi) = \begin{bmatrix}\cos\psi & -\sin\psi & 0\\ \sin\psi & \cos\psi & 0\\ 0 & 0 & 1\end{bmatrix} \qquad R_\phi(\phi) = \begin{bmatrix}\cos\phi & 0 & \sin\phi\\ 0 & 1 & 0\\ -\sin\phi & 0 & \cos\phi\end{bmatrix}. \tag{8}$$

If the antenna is pointed in direction $\mathbf{u}$, then the antenna pattern in the direction $\mathbf{u}_1$ is $A(R_\phi(-\phi)R_\psi(-\psi)\mathbf{u}_1)$. If the direction $\mathbf{u}_1$ is close to $\mathbf{u}$, with angles $\phi_1 = \phi + \Delta\phi$ and $\psi_1 = \psi + \Delta\psi$, then the resulting rotated direction vector is given by (to second order)

$$R_\phi(-\phi)R_\psi(-\psi)\mathbf{u}_1 \approx \begin{bmatrix}\Delta\phi - (\Delta\psi^2/2)\sin\phi\cos\phi\\ \Delta\psi\sin\phi + \Delta\phi\Delta\psi\cos\phi\\ 1-\Delta\phi^2/2 - (\Delta\psi^2/2)\sin^2\phi\end{bmatrix}. \tag{9}$$

For the following, we define

$$R(\phi,\psi) = R_\psi(\psi)R_\phi(\phi)\,. \tag{10}$$

The target is assumed to be moving at a constant velocity v m/s with respect to the antenna, yielding a doppler shift of

$$f = 2vf_0/c\,, \tag{11}$$

where f_0 is the carrier frequency. Assuming a transmitted complex envelope of $\tilde{s}_T(t)$ and a direction (ϕ,ψ) for the antenna, the complex envelope of the received signal may be modeled as

$$\tilde{r}(t,\phi,\psi) = \tilde{\beta}\,\frac{4}{c^2\tau^2}\,A^2(R^T(\phi,\psi)\mathbf{u}_1)\tilde{s}_T(t-\tau)e^{j2\pi ft} + \tilde{w}(t,\phi,\psi)\,, \tag{12}$$

where the target is assumed to be in direction $\mathbf{u}_1$, and $\tilde{w}$ is receiver noise modeled as additive complex white Gaussian noise with spectral power N_0. Let E_T be the energy in the signal $\tilde{s}_T(t)$. For fixed (ϕ,ψ), a sufficient statistic for a target at delay doppler coordinates (τ,f) is

$$r_1(\phi,\psi,\tau,f) = \frac{1}{\sqrt{E_T}}\int \tilde{r}(t,\phi,\psi)\tilde{s}_T(t-\tau)e^{-j2\pi ft}\,dt\,. \tag{13}$$

Under our model, the random variable $r_1(\phi,\psi,\tau,f)$ is complex Gaussian with zero mean and variance

$$\sigma_r^2(\phi,\psi) = \sigma^2 E_T\,\frac{16}{c^4\tau^4}\,|A(R^T(\phi,\psi)\mathbf{u}_1)|^4 + N_0\,. \tag{14}$$

If in fact the target is at delay doppler coordinates (τ_1, f_1), then the signal part of the variance has an extra multiplier equal to the squared magnitude of the complex ambiguity function of the transmitted signal evaluated at $(\tau - \tau_1, f - f_1)$; denote this variance by $\sigma_r^2(\phi, \psi, \tau, f)$. The complex ambiguity function is defined by

$$\chi(\tau, f) = \frac{1}{E_T} \int \tilde{s}_T(t) \tilde{s}_T^*(t - \tau) e^{-j2\pi ft} dt \; . \tag{15}$$

Examining the form of the variance in (14) motivates one strategy for tracking. In order to estimate the direction of the target, the antenna is pointed in a finite set of directions, $\{(\phi(k), \psi(k)), \; k = 1, 2, \cdots, K\}$, and identical signals $\tilde{s}_T(t)$ are transmitted. For each direction, the quantity defined in (13) is computed for a range of (τ, f). Estimates of (τ, f) and (ϕ, ψ) are then obtained.

5. TRACKING LIKELIHOOD

The previous section derives a detailed model for the data available from the tracking radar. In this section, the likelihood for the data is derived. This likelihood is connected to a prior likelihood on the target's motion to yield the tracking likelihood. Combined with the results of this section and the previous section on high resolution data, this yields a joint likelihood on the tracking and image data. This joint likelihood may then be examined to yield quantitative performance measures.

Assume that the data from (12) are available for a range of angles (ϕ, ψ). While a continuous range of angles is not available in practice, this is a good starting point for the analysis. Also assume that

$$\int_0^{2\pi} \int_0^{\pi} |A(\phi, \psi)|^4 \sin \phi d\phi d\psi = G_T^2 < \infty \; . \tag{16}$$

Then a sufficient statistic is given by the integral of the left side of (13) times the antenna function:

$$r_2(\phi, \psi, \tau, f) = \frac{1}{G_T} \int_0^{2\pi} \int_0^{\pi} r_1(\phi', \psi', \tau, f) A^{2*}(R^T(\phi', \psi')\mathbf{u}(\phi, \psi)) \sin \phi' d\phi' d\psi' \; . \tag{17}$$

From (12), if (ϕ, ψ, τ, f) corresponds to the actual target, then r_2 is complex Gaussian, zero mean with variance

$$\sigma_2^2 = \sigma^2 E_T \frac{16}{c^4 r^4} G_T^2 + N_0 \; . \tag{18}$$

This implies that the likelihood for the data is

$$l(r_2; \phi, \psi, \tau, f) = -\ln \sigma_2^2 - \frac{|r_2|^2}{\sigma_2^2} \; . \tag{19}$$

A maximum likelihood direction estimate is then found by maximizing (19) over ϕ and ψ. The joint maximum likelihood estimates for direction and delay-doppler coordinates is found by maximizing (19) over all four variables. Furthermore, the optimal performance can be determined in terms of this likelihood. That is, the Cramer-Rao bounds on the estimates are obtained from the expected value of the Hessian of the likelihood with respect to the variables.

As mentioned above, in practice only a finite number of angles is available. Our model is of a target that does not fluctuate during the illumination of the target over these angles ($\tilde{\beta}$ in (12) is not time-varying). The sufficient statistic for this problem is given by

$$r_3(\phi,\psi,\tau,f) = \frac{1}{G_T(\phi,\psi)} \sum_{k=1}^{K} r_1(\phi(k),\psi(k),\tau,f) A^{2*}(R^T(\phi(k),\psi(k))\mathbf{u}(\phi,\psi)) \, , \tag{20}$$

where $G_T(\phi,\psi)$ is a discrete approximation to G_T and equals

$$G_T^2(\phi,\psi) = \sum_{k=1}^{K} |A(R^T(\phi(k),\psi(k))\mathbf{u}(\phi,\psi))|^4 \, . \tag{21}$$

If (ϕ,ψ,τ,f) correspond to the actual target, then r_3 is complex $N(0,\sigma_3^2)$, where

$$\sigma_3^2(\phi,\psi) = \sigma^2 E_T \frac{16}{c^4\tau^4} G_T^2(\phi,\psi) + N_0 \, . \tag{22}$$

The likelihood is

$$l(r_3;\phi,\psi,\tau,f) = -\ln\sigma_3^2(\phi,\psi) - \frac{|r_3(\phi,\psi,\tau,f)|^2}{\sigma_3^2(\phi,\psi)} \, . \tag{23}$$

The direct maximization of (23) and the prediction of performance, while tractable, is rather involved. For the purposes of this paper, a simplified analysis is used to predict performance.

6. APPROXIMATE PERFORMANCE

While equation (23) may be maximized directly, more straightforward analysis based on expanding the antenna function in the neighborhood of the peak may yield performance guidelines. An outline for such an approach is given here.

First, the antenna function may be written as

$$A(\mathbf{u}) = \int_{\Omega} g(\mathbf{r}) \exp[j\frac{2\pi}{\lambda} < \mathbf{u},\mathbf{r} >] d\mathbf{r} \, , \tag{24}$$

where $g(\mathbf{r})$ is the illumination function, Ω is the antenna aperture, and $< \cdot,\cdot >$ is an inner product in $\mathbf{R}^3$. Assume that $g(\mathbf{r})$ is concentrated on the $x-y$ plane (denote it by $g(x,y)$) and is symmetric in the sense that

$$\int\int g(x,y)x\,dxdy = \int\int g(x,y)y\,dxdy = 0 \, . \tag{25}$$

The peak of $A(\mathbf{u})$ is at $\mathbf{u} = [0\ 0\ 1]^T$ and equals $\int\int g(x,y)dxdy$. In the neighborhood of the peak, using (9) and keeping only terms to second order,

$$\begin{aligned} A(R(\phi,\psi)\mathbf{u}_1) &\approx A([0\ 0\ 1]^T) - \frac{2\pi^2}{\lambda^2} \mathrm{Re}\left[\int\int g(x,y)(x\Delta\phi + y\Delta\psi\sin\phi)^2 dxdy\right] \\ &= A([0\ 0\ 1]^T)\left(1 - \frac{2\pi^2}{\lambda^2}\left[\sigma_x^2\Delta\phi^2 + 2\rho\sigma_x\sigma_y\Delta\phi\Delta\psi\sin\phi + \sigma_y^2\Delta\psi^2\sin^2\phi\right]\right), \end{aligned} \tag{26}$$

where

$$\sigma_x^2 = \frac{1}{\int\int g(x,y)dxdy} \int\int x^2 g(x,y)dxdy \, ,$$

and similarly for $\rho\sigma_x\sigma_y$ and σ_y^2. Using (26), the following approximation holds

$$|A(R(\phi,\psi)\mathbf{u}_1)|^4 = |A([0\ 0\ 1]^T)|^4\left(1 - \frac{8\pi^2}{\lambda^2}\left[\sigma_x^2\Delta\phi^2 + 2\rho\sigma_x\sigma_y\Delta\phi\Delta\psi\sin\phi + \sigma_y^2\Delta\psi^2\sin^2\phi\right]\right). \qquad (27)$$

This may be used in (23) to simplify the computations.

It should be noted that there is a prior on the direction of the target from the previous step of the tracking algorithm. Using a dynamic model for the target as in [12] there is a likelihood relating the high resolution radar data and the tracking radar data.

7. CONCLUSIONS

This paper discusses a detailed likelihood model for joint tracking and high resolution radar data. This likelihood may be used for target recognition when combined with a motion likelihood as in [12]. Simulated range profiles are shown. In an actual implementation, there are two possible uses for the simulated profiles shown. One possibility is that the profiles may be precomputed for a wide range of orientations of the target. The estimation of orientation then consists of a search throughout the set of orientations (guided by the likelihood on the target orientation). The disadvantage to this approach is the large number of profiles which must be stored for quick access. A second possibility is that the range profiles could be computed on-line. The simple models presented here might be used in such a scheme. One advantage of this approach is that arbitrarily small changes of orientation may be simulated. The disadvantage is the computation time relative to a search. We are implementing a target recognition algorithm on a DEC/mpp 12000 and plan to compare the two alternatives. The algorithm is based on the jump-diffusion algorithm presented in [11].

8. REFERENCES

References

1. M. I. Skolnik, *Introduction to Radar Systems, Second Edition,* McGraw-Hill, New York, 1980.
2. F. E. Nathanson, J. P. Reilly, and M. N. Cohen, *Radar Design Principles: Signal Processing and the Environment,Second Edition,* McGraw-Hill, New York, 1991.
3. D. R. Wehner, *High Resolution Radar,* Artech House, Dedham, MA., 1987.
4. D. L. Mensa, *High Resolution Radar Imaging,* Artech House, Norwood, MA, 1981.
5. A. W. Rihaczek, *Principles of High-Resolution Radar,* Peninsula Publishing, Los Altos, CA, 1985.
6. H. L. Van Trees, *Detection, Estimation, and Modulation Theory, Part III,* John Wiley and Sons, New York, 1971.
7. D.L. Snyder, J.A. O'Sullivan, and M.I. Miller, "The Use of Maximum-Likelihood Estimation for Forming Images of Diffuse Radar-Targets from Delay-Doppler Data," *IEEE Transactions on Information Theory*, vol. IT-35, pp. 536-548, 1989.
8. P. Moulin, J. A. O'Sullivan, and D. L. Snyder, "A Method of Sieves for Multiresolution Spectrum Estimation and Radar Imaging," *IEEE Transactions on Information Theory,* vol. IT-38, pp. 801-813, March 1992.
9. B. Friedland, *Control System Design: An Introduction to State-Space Methods,* McGraw-Hill, New York, 1986.

10. J. A. O'Sullivan, K. C. Du, R. S. Teichman, M. I. Miller, D. L. Snyder, and V. C. Vannicola, "Radar Target Recognition Using Shape Models," *Proceedings 30th Allerton Conference on Communication, Control, and Computing*, pp. 515-523, University of Illinois, Urbana, IL, October 1992.

11. U. Grenander and M. I. Miller, "Representations of Knowledge in Complex Systems," *Journal of the Royal Statistical Society*, submitted February 1992.

12. A. Srivastava, N. Cutaia, M. I. Miller, J. A. O'Sullivan, and D. L. Snyder, "Multi-Target Narrowband Direction Finding and Tracking Based on Motion Dynamics," *submitted for publication. Electronic Systems and Signals Research Monograph*, Department of Electrical Engineering, Washington University, St. Louis, MO 63130, July 1992.

13. A. Srivastava, M. I. Miller, and U. Grenander, "Jump-Diffusion Processes for Object Tracking and Direction Finding," *Proceedings 29th Annual Allerton Conference on Communication, Control, and Computing*, pp. 563-570, University of Illinois, Urbana-Champaign, 1991.

Figure 1. Target and one range profile. The target is illuminated from the left.

Figure 2. Twenty range profiles of the target from Figure 1 at one degree increments.

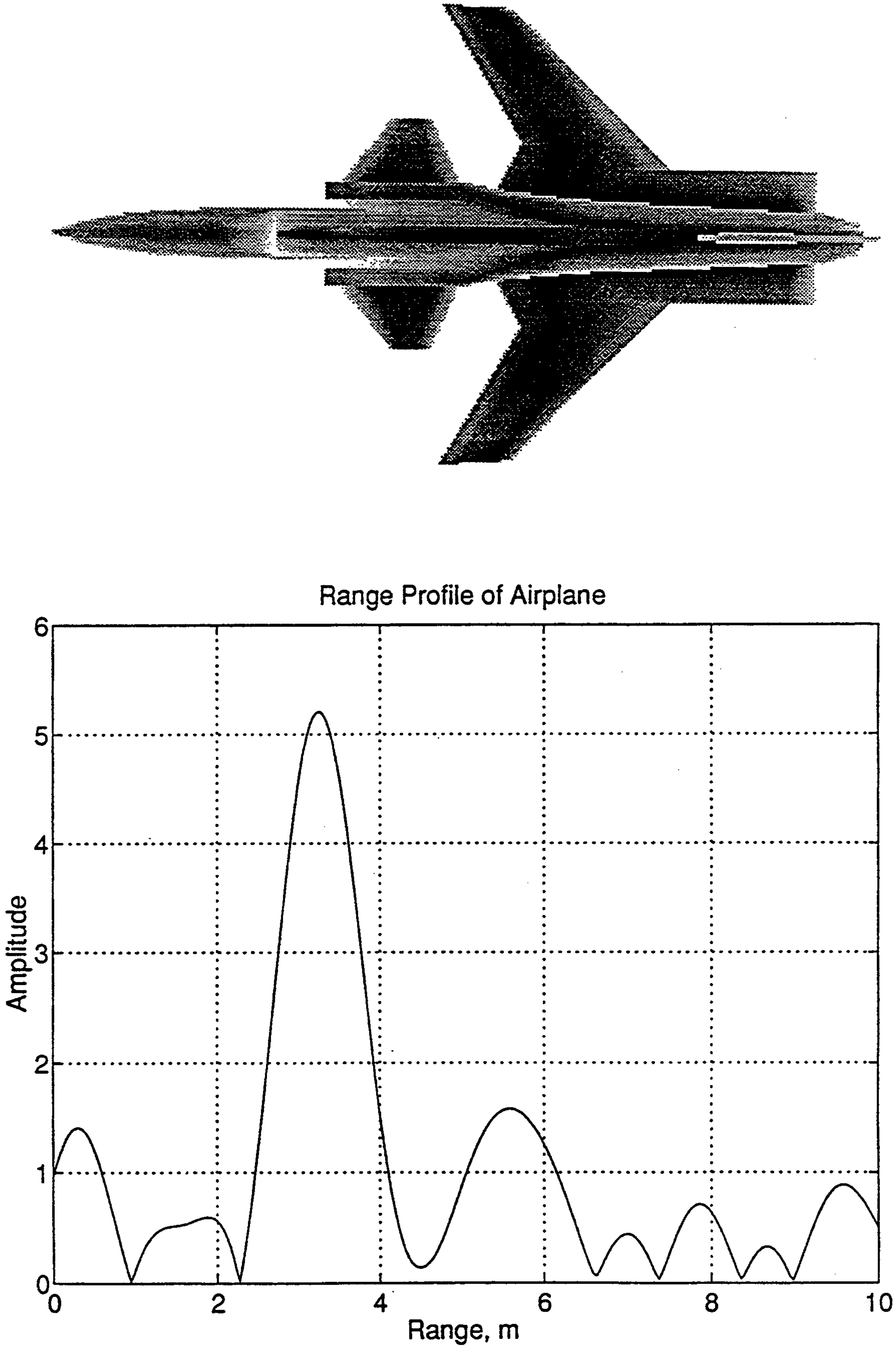
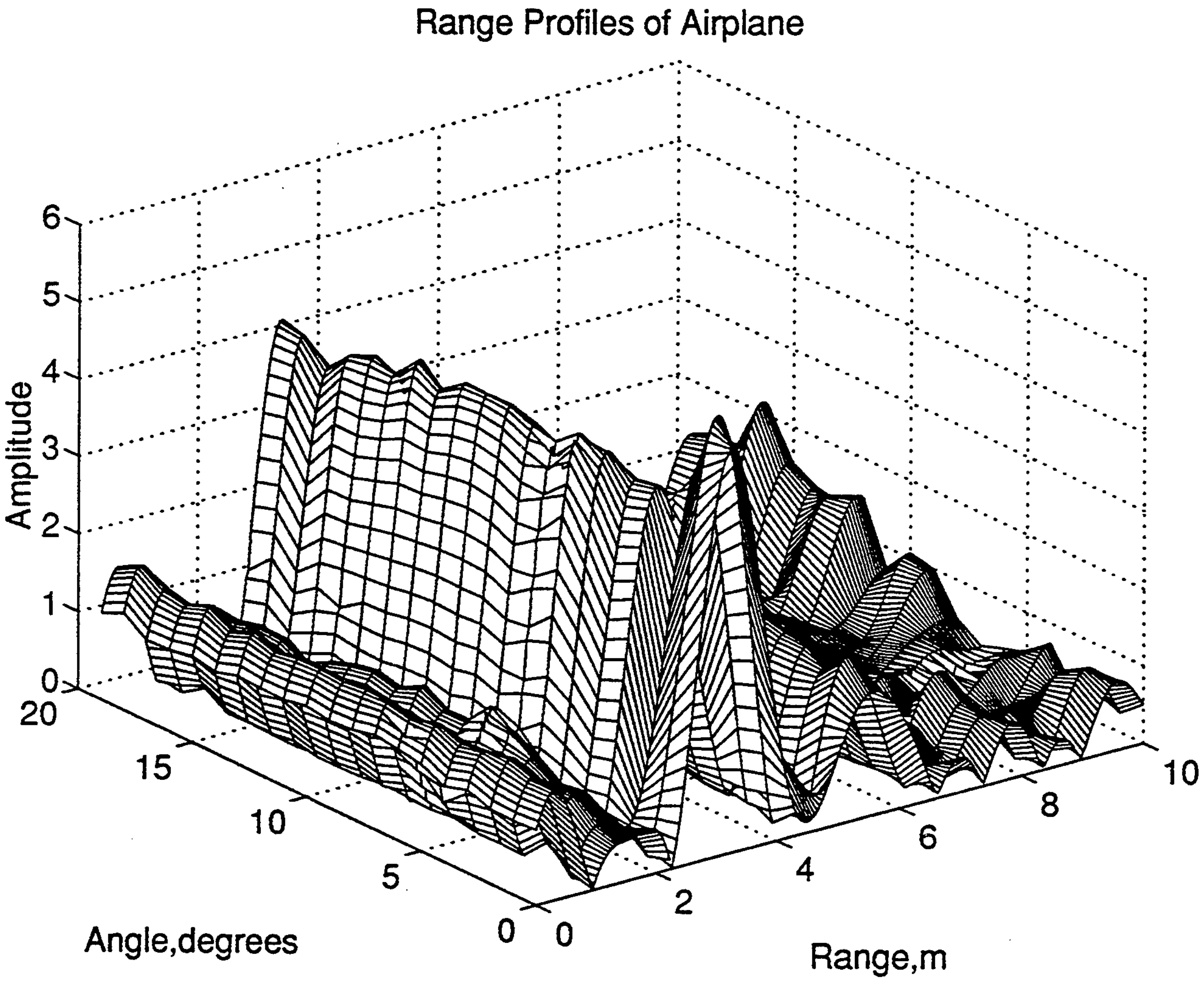


Figure 1. Target and one range profile. The target is illuminated from the left.


Range Profiles of Airplane
Amplitude
0
1
2
3
4
5
6
20
15
10
5
0
Angle,degrees
0
2
4
6
8
10
Range,m

TRACKING USING A RANDOM SAMPLING ALGORITHM

J.A. O'SULLIVAN, M.I. MILLER, A. SRIVASTAVA, D.L. SNYDER

Electronic Systems and Signals Research Laboratory, Department of Electrical Engineering, Campus Box 1127, Washington University, St. Louis, MO 63130, USA.

E-mail: *jao@saturn.wustl.edu*, Telephone: (314) 935-4173

Abstract. A random sampling algorithm for nonlinear state estimation is formulated. Special cases of the algorithm are presented. A continuous Markov process is defined, along with a measurement distribution. From these, the posterior distribution on the states is derived. The sampling algorithm generates realizations from this distribution. A parallel implementation of the algorithm is presented. An application discussed is the problem of tracking radar targets using data from multiple sensors.

Key Words. Target tracking, nonlinear systems, state estimation, random processes, stochastic systems.

1. INTRODUCTION

The main result in this paper is a random sampling algorithm for state estimation for Markov random processes. This algorithm is based on recent results in the statistics literature (Grenander and Miller, 1991; Geman and Geman, 1984; Geman and Hwang, 1986) and on multitarget tracking (Srivastava, *et al.*, 1991; Srivastava, *et al.*, 1992). Since the algorithm samples from the posterior distribution and is ergodic under appropriate assumptions, arbitrary functions of the state may be estimated. For example, the sample average converges to the posterior mean. Annealing can be used with the sampling algorithm to produce the maximum a posteriori probability (MAP) estimate of the state. A parallel implementation of the proposed algorithm having block nearest neighbor connectivity is presented in Section 3.

When the number of signals contributing to measurements is unknown, this sampling algorithm is extended to a jump-diffusion algorithm (Grenander and Miller, 1991; Srivastava, *et al.*, 1991; Srivastava, *et al.*, 1992) which also estimates the number of signals. This is applicable to multitarget tracking where the number of targets is unknown, but the dynamics of possible targets are known. This approach is very different from approaches discussed in the literature (see, for example, Bar-Shalom (1990), Bar-Shalom and Fortmann (1988), and Blackman (1986)) and would work most effectively for a modest number of targets.

2. MODEL DEFINITION AND RANDOM SAMPLING ALGORITHM

Assume there is a continuous valued Markov process underlying the data. Let the state $x(t) \in \mathbf{R}^n$ for all $t \geq 0$. The probability density function on $x(0)$ is known and given by the Gibbs' density

$$p(x;0) = \frac{1}{Z(0)} e^{-E(x;0)} ,$$

where $E(x;t)$ is an energy function, and $Z(t)$ is the partition function

$$Z(t) = \int e^{-E(x;t)} dx .$$

This work was supported in part by Rome Laboratory under contract F30602-92-C-0004, in part by the Office of Naval Research under contract 59922, and in part by the National Science Foundation under grant MIP-9101991.

The conditional density function on $x(t)$ given $x(t')$ is given by

$$p(x;t|x';t') = \frac{1}{Z(t,t')} e^{-E(x;t|x';t')} ,$$

where $E(x;t|x';t')$ is called a conditional energy function. It is assumed that $Z(t,t')$ does not depend on x'. The measurable random vector $y(t) \in \mathbf{R}^p$ depends only on the state at the same time, $x(t)$, and has conditional density function

$$p(y|x;t) = \frac{1}{Z_2(t)} e^{-E_2(y|x;t)} ,$$

where, by assumption, $Z_2(t)$ does not depend on x. Conditioned on the state process, the outputs at different times are independent. All energy functions are assumed to be differentiable in x, x', and y with Lipschitz continuous derivatives.

Now an algorithm is developed to sample the posterior density on the states given a finite set of measurements. Suppose the output is sampled at times $0 = t_0 < t_1 < t_2 < \cdots < t_{N-1}$. Then the posterior density on the states at those same times has energy function

$$E(\mathbf{X}|\mathbf{Y}) = E(x_0;0) + E_2(y_0|x_0;0)$$
$$+ \sum_{k=1}^{N-1} E(x_k;t_k|x_{k-1};t_{k-1}) + E_2(y_k|x_k;t_k) ,$$

where $x_k = x(t_k)$, $y_k = y(t_k)$, and

$$\mathbf{X}^T = [x_0^T \; x_1^T \cdots x_{N-1}^T]$$
$$\mathbf{Y}^T = [y_0^T \; y_1^T \cdots y_{N-1}^T]$$

(the superscript T denotes transpose).

Following (Miller, *et al.*, 1991; Srivastava, *et al.*, 1991) a Langevin stochastic differential equation to sample from this posterior is given by

$$d\xi(t) = -\nabla_{\mathbf{X}} E(\xi(t)|\mathbf{Y})dt + \sqrt{2}d\mathbf{w}(t) ,$$

where $\xi(t) \in \mathbf{R}^{nN}$, and $\mathbf{w}(t)$ is an nN dimensional standard Wiener process. The limit distribution for this differential equation has energy function $E(\mathbf{X}|\mathbf{Y})$ (Ikeda and Watanabe, 1989). Samples of functions of the posterior may be obtained from this process.

The values of the posterior probabilities of the states at other times may be sampled as well. To do so, two terms are added to the energy function and one is removed. For example, if the state at time t' is of interest, and $t_k < t' < t_{k+1}$, then the two energies

$$E(x';t'|x_k;t_k) + E(x_{k+1};t_{k+1}|x';t')$$

are added and the energy

$$E(x_{k+1};t_{k+1}|x_k;t_k)$$

is removed.

If the MAP estimate of the state is desired, the algorithm above may be annealed by multiplying the Wiener process by a temperature parameter. The new sampling algorithm has the form

$$d\xi(t) = -\nabla_{\mathbf{X}} E(\xi(t)|\mathbf{Y})dt + \sqrt{2}T(t)d\mathbf{w}(t) .$$

Standard methods may be used to determine the cooling schedule for $T(t)$ (Geman and Geman, 1984). For the remainder of this paper, no annealing is performed ($T(t) = 1$).

It is not required that $p < n$. In fact, one application of the methods discussed here is in image-while-track algorithms where the image data are fed back to the tracking algorithm. The recent paper (O'Sullivan, *et al.*, 1992) derives the likelihood function for multisensor data for the problem of simultaneously imaging and tracking. The algorithm in this paper can use that likelihood for tracking.

3. PARALLEL ALGORITHM DECOMPOSITION

The gradient computation required for the random sampling algorithm has blocks which are the gradients with respect to the individual vectors x_k. These are given by

$$\nabla_{x_k} E(\mathbf{X}|\mathbf{Y}) = \nabla_{x_k} E(x_k;t_k|x_{k-1};t_{k-1})$$
$$+ \nabla_{x_k} E(x_{k+1};t_{k+1}|x_k;t_k) + \nabla_{x_k} E_2(y_k|x_k;t_k) .$$

Thus, in the sampling algorithm the variables corresponding to x_k need only be connected to the variables corresponding to x_{k-1} and x_{k+1} in addition to being connected to themselves and y_k. The first term in the expression for $\nabla_{x_k} E(\mathbf{X}|\mathbf{Y})$ measures how well x_k corresponds to x_{k-1}. The second term measures the ability of x_k to predict x_{k+1}. The third term measures the correspondence between x_k and y_k.

A parallel architecture for performing the sampling would have N cells. Define the function g_k by

$$g_k(x_k, x_{k+1}, x_{k-1}, y_k) = \nabla_{x_k} E(\mathbf{X}|\mathbf{Y}) .$$

Denote by ξ_k the entries of ξ corresponding to x_k. Similarly let $w_k(t)$ be the entries of $\mathbf{w}(t)$ corresponding to x_k; so $w_k(t)$ is a standard n dimensional Wiener process. Then the k^{th} cell computes

$$d\xi_k(t) =$$
$$-g_k(\xi_k(t), \xi_{k+1}(t), \xi_{k-1}(t), y_k)dt + \sqrt{2}dw_k(t) .$$

The implementation of this cell would be an extension of the ideas presented in (Miller, *et al.*, 1991).

4. EXAMPLE: LINEAR MODEL

Suppose that $x(t)$ satisfies the linear time-invariant state equation

$$dx = Axdt + Bdu ,$$

where $u(t)$ is a standard m dimensional Wiener process. Assume that $x(0)$ is a zero mean Gaussian random vector with covariance matrix $V(0)$. Suppose that the measurements are given by

$$y(t_k) = Cx(t_k) + v_k ,$$

where $\{v_k\}$ is a sequence of independent, p dimensional, Gaussian, random vectors with zero mean and covariance matrix R_k. Then

$$E(x_k; t_k | x_{k-1}; t_{k-1}) = \frac{1}{2}(x_k - e^{A(t_k - t_{k-1})}x_{k-1})^T \cdot$$
$$K^{-1}(t_k - t_{k-1})(x_k - e^{A(t_k - t_{k-1})}x_{k-1}) ,$$

where $K(\tau) = \int_0^\tau e^{At}BB^T e^{A^T t}\,dt$.

For details, see Melsa and Sage (1973, pg 255). The energy function for y_k is

$$E(y_k | x_k; t_k) = \frac{1}{2}(y_k - Cx_k)^T R_k^{-1}(y_k - Cx_k) .$$

For this case, the functions g_k are linear in their arguments and are given by

$$g_k(x_k, x_{k+1}, x_{k-1}, y_k) =$$
$$K^{-1}(t_k - t_{k-1})e^{A(t_k - t_{k-1})}(e^{-A(t_k - t_{k-1})}x_k - x_{k-1})$$
$$- e^{A^T(t_{k+1} - t_k)}K^{-1}(t_{k+1} - t_k)(x_{k+1} - e^{A(t_{k+1} - t_k)}x_k)$$
$$- C^T R_k^{-1}(y_k - Cx_k) .$$

The expression for g_k in this case has an elegant interpretation. This first term is driven by the backwards prediction error of x_{k-1} given x_k. The second term is driven by the forward prediction error of x_{k+1} given x_k. The third term is driven by the output estimation error of y_k given x_k.

If $t_k - t_{k-1} = \tau$ is a constant and $R_k = R$ is constant for all $k = 1, 2, ..., N-1$, then all of the interior computation cells are identical, simplifying the computational architecture. The implementation of the random sampling algorithm for this linear case is particularly easy. Note that the time-invariance assumption did not play a significant role in the derivation and can be removed. The linearity and Gaussian assumptions are crucial for the final form here. For an application to tracking, see Srivastava, *et al.* (1992).

The usual approach for the linear case is to use a smoothing filter to obtain a minimum mean squared error (MMSE) estimate for the states. The MMSE estimate equals the MAP estimate, and thus may be obtained from the algorithm above by annealing. In fact, since the posterior is unimodal (it's Gaussian), the temperature may be set to 0 and the annealing algorithm gives a simple gradient descent algorithm to compute the estimate. In practice, one would not use the proposed algorithm to compute the MMSE estimate. As described by Tretter (1976, pg. 386), the MMSE estimate may be found in two passes through the data. In the first pass, the optimal filtered estimates,

$$\hat{x}_{k|k} = E\{x_k | y_l,\ 0 \le l \le k\} ,$$

are computed and saved. Also saved are the covariance matrices for the one step ahead state prediction error and the state filtering error. This first pass starts with $\hat{x}_{0|0}$ and works forward in time to $\hat{x}_{N-1|N-1}$. The second pass works backward in time, computing $\hat{x}_{N-2|N-1}$ and continuing to $\hat{x}_{0|N-1}$. Also computed in this second pass are the smoothing error covariance matrices. Since the posterior is Gaussian, these estimates and covariance matrices completely determine the marginal density functions on x_k given the measurements. The covariance between these estimates is more difficult to determine, but may be found from the innovations representation of the estimates. The sampling algorithm would allow the computation of expected values of functions of the entire sequence $\mathbf{X}$, given $\mathbf{Y}$, if such functions are desired.

5. EXAMPLE: NONLINEAR DISCRETE TIME MODEL

The example considered here is a nonlinear system whose state equation is linear in the input. Suppose a nonlinear discrete time system has state equation

$$x_{k+1} = f(x_k) + u_k , \qquad (1)$$

where $x_k \in \mathbf{R}^n$, $u_k \in \mathbf{R}^n$, and $f(\cdot)$ is a smooth function. The output equation is given by

$$y_k = h(x_k) + v_k , \qquad (2)$$

where y_k and v_k are in $\mathbf{R}^p$, and $h(\cdot)$ is smooth. Assume that the probability density function on x_0 is known. Let u_k be an independent and identically distributed (i.i.d.) sequence of Gaussian random vectors with zero mean and positive definite covariance matrix Q; u_k is independent of x_k. Let v_k be an i.i.d. sequence of Gaussian random vectors with zero mean and positive definite covariance matrix R; this sequence is independent of the sequence of inputs u_k and the states x_k.

Given x_k, x_{k+1} is conditionally Gaussian with mean $f(x_k)$ and covariance matrix Q. The conditional energy function is

$$E(x_k; k|x_{k-1}; k-1) = \frac{1}{2}(x_k - f(x_{k-1}))^T Q^{-1}(x_k - f(x_{k-1})) .$$

The output energy function is

$$E(y_k|x_k; k) = \frac{1}{2}(y_k - h(x_k))^T R^{-1}(y_k - h(x_k)) .$$

Combining these two energy functions, the gradient needed for the sampling algorithm may be found to be

$$\begin{aligned} g_k(x_k, x_{k+1}, x_{k-1}, y_k) &= Q^{-1}(x_k - f(x_{k-1})) \\ &\quad - \nabla_x f^T(x_k) Q^{-1}(x_{k+1} - f(x_k)) \\ &\quad - \nabla_x h(x_k)^T R^{-1}(y_k - h(x_k)) . \end{aligned}$$

Note that this computation has a similar interpretation to the linear case. The first term is driven by the error between x_k and its estimate given x_{k-1}. The second term is driven by the forward prediction error between x_{k+1} and its estimate given x_k. The third term is driven by the output estimation error between y_k and its estimate given x_k.

6. CONCLUSIONS

A method for random sampling of the states of Markov processes has been proposed. This algorithm can be extended to a multitarget tracking algorithm using methods from (Srivastava, *et al.*, 1991; Srivastava, *et al.*, 1992). Special cases and examples were introduced, including linear state space models and nonlinear state space models with additive Gaussian noise.

Recent results derive posterior distributions for a multisensor track-while-image algorithm (O'Sullivan, *et al.*, 1992). A simplified version of this posterior fits into the model above. This application is of interest to the authors, including the use of this algorithm in object identification problems. The authors are presently developing an implementation on a DEC/MPP 12000 for radar target identification purposes.

7. REFERENCES

Bar-Shalom, Y., Ed. (1990). *Multitarget Multisensor Tracking: Advanced Applications.* Artech House, Norwood, MA.

Bar-Shalom, Y., and T. E. Fortmann (1988). *Tracking and Data Association.* Academic Press, New York.

Blackman, S.S. (1986). *Multiple Target Tracking with Radar Applications.* Artech House, Dedham, MA.

Geman, S., and D. Geman (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Trans. Pattern Anal. Mach. Intel.*, **6,** 721-741.

Geman, S., and C.-R. Hwang (1986). Diffusions for global optimization. *SIAM J. Cont. Optim.*, **24,** 1031-1043.

Ikeda, N., and S. Watanabe (1989). *Stochastic Differential Equations and Diffusion Processes, Second Edition.* North-Holland, Amsterdam.

Melsa, J.L., and A. P. Sage (1973). *An Introduction to Probability and Stochastic Processes.* Prentice-Hall, Englewood Cliffs, NJ.

Grenander, U. and M. I. Miller (1991). Representations of knowledge in complex systems. *J. Royal Stat. Soc.*, in review.

Miller, M.I., B. Roysam, K. Smith, and J.A. O'Sullivan (1991). Representing and computing regular languages on massively parallel networks. *IEEE Trans. Neural Networks,* **2,** 56-72.

O'Sullivan, J.A., K.C. Du, R.S. Teichman, M.I. Miller, D.L. Snyder, and V.C. Vannicola (1992). Radar target recognition using shape models. *Proc. 30th Allerton Conf. Comm., Cont., and Comp.,* Univ. Illinois, Urbana, IL, 515-523.

Srivastava, A., N. Cutaia, M.I. Miller, J.A. O'Sullivan, and D.L. Snyder (1992). Multi-target narrowband direction finding and tracking based on motion dynamics. *Proc. 30th Allerton Conf. Comm., Cont., and Comp.,* Univ. Illinois, Urbana, IL, 279-288.

Srivastava, A., M.I. Miller, and U. Grenander (1991). Jump-diffusion processes for object tracking and direction finding. *Proc. 29th Allerton Conf. Comm., Cont., and Comp.,* Univ. Illinois, Urbana, IL, 563-570.

Tretter, S.A. (1976). *Introduction to Discrete-Time Signal Processing.* John Wiley and Sons, New York.

JUMP-DIFFUSION PROCESSES FOR AUTOMATED TRACKING-TARGET RECOGNITION *

Michael I. Miller, Robert Teichman, Anuj Srivastava,
Joseph A. O'Sullivan, Donald L. Snyder

Electronic Signals and Systems Research Laboratory
Washington University, St. Louis, Missouri 63130

*Invited Paper to the 1993 Conference on Information Sciences and Systems

1 Recognition Via Deformable Templates

A fundamental task in the representation of complex dynamically changing scenes involving rigid targets is the construction of models that incorporate both the variability of orientation, range, and object number, as well as the precise rigid structure of the objects in a mathematically precise way. The global deformable shape models introduced in Grenander's general pattern theory [1] extended to parametric representations of arbitrary unknown model order [2, 3] are intended to do this. This becomes the basis for our approach.

There are various kinds of variability and uncertainty inherent to data obtained via remote sensing via high resolution and tracking radars. The first and foremost variability is associated with the conformations of the rigid bodies: orientations, scales, and position. To accommodate this type of variability we use global templates which are made flexible via the introduction of basic transformations involving both rigid motions of translation and rotation, as well as non-rigid motions such as scale. As these transformations correspond to parameters which are parameters in 3-D Euclidean space, as well as angles in the continuum, the transformations are performed using continuous stochastic gradient search. The *second* kind of variability is associated with the model order, or parametric dimension. In any scene there may be multiple, variable numbers of targets, with tracks of variable length and the target number unknown apriori.

We take a Bayesian approach by defining a prior density on the scenes of targets. The prior coupled to the sensor data likelihood gives the Bayes posterior. The conformation is selected to be consistent with the data in the sense that scenes of high probability under the posterior distribution are selected. Our method for generating candidate conformations is to sample from the posterior. For this a new class of random sampling algorithms is used based on jump-diffusion dynamics, introduced in Grenander and Miller [2, 3, 4] which visit candidate solutions according to the posterior density. The original motivation for introducing jump-diffusions is to accommodate the very different continuous and discrete components of the discovery process. Given a conformation associated with a target type, or group of targets, the problem is to track and identify the orientation, translation and scale parameters accommodating the variability manifest in the viewing of each object type. For this, the parameter space is sampled using Langevin stochastic differential equations in which the state vector continuously winds through the translation-rotation-scale space following gradients of the posterior. The second distinct part of the sampling process supports the notion that the recognition part of the deduction is associated with choosing the target types. The deduction algorithm goes through multiple stages of hypothesis during which the airplane types are being discovered, and some subset of the scene may be only partial "recognized". This is accommodated by defining the second transformation type which jumps between different object types, where a jump may correspond to the hypothesis of a new object in the scene, or a "change of mind" about an object type. The jump intensities are governed by the posterior density, with the process visiting conformations of higher probability for longer exponential times, and the diffusion equation governing the dynamics between jumps.

2 Global Templates

We use the global shape models and pattern theoretic approach introduced by Grenander [5, 1]. As the basic building blocks of the hypotheses define the set of generators $\mathcal{G}$, the targets placed at the origin of the *reference coordinates* at a fixed orientation, position, and unit scale.

The fundamental variability in target spaces is accommodated by applying the transformations $T(\vec{\phi}), T(\vec{p}), T(\vec{s})$ to the templates $\mathcal{G}$ according to

$$T(\vec{\phi}) \; : \; \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi_1 & sin\phi_1 \\ 0 & -sin\phi_1 & cos\phi_1 \end{bmatrix} \times \quad (1)$$

$$\begin{bmatrix} cos\phi_2 & 0 & sin\phi_2 \\ 0 & 1 & 0 \\ sin\phi_2 & 0 & cos\phi_2 \end{bmatrix} \begin{bmatrix} cos\phi_3 & sin\phi_3 & 0 \\ -sin\phi_3 & cos\phi_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$T(\vec{p}) \; : \; \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \rightarrow \begin{bmatrix} x_1 + p_1 \\ x_2 + p_2 \\ x_3 + p_3 \end{bmatrix} \quad (2)$$

$$T(s) \; : \; \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \rightarrow \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & s \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} , \quad (3)$$

where $\vec{\phi}$ is the triple of rotation angles associated with the rotation of the viewing sphere about each object, $\vec{p}$ are the translation parameters in R^3, and the s is the scale parameter in R_+. These parameterized transformations operate on the template targets of $\mathcal{G}$ generating the full target space.

Shown in Figure 1 is one of the 3-D ideal targets, used for all of the simulations below. The left panel shows the target at the origin, the right panel showing the result of applying one of the transformations.

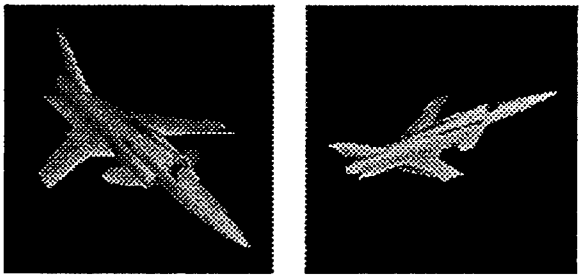


Figure 1: 3-D target at the origin (left panel) and after applying transformations (right panel)

2.1 The Parametric Space

Now the parametric space parameterizing the Bayes posterior on the patterns becomes the set of parameters specifying the similarity transformations, as well as the airplane type. Associate with each target or generator $g \in \mathcal{G}$ the parameter vector $\vec{x} \in \mathcal{O}(3) \times R^3 \times R_+ \times \mathcal{A}$, where $|\mathcal{A}| = |\mathcal{G}|$ the number of different target types.

A pattern will be constructed from multiple targets with varying track lengths. In [6] we have described the multi-target scenario. Here we focus on single target scenes. We are interested in tracking and recognition in "hostile/non-cooperative" environments in which the objects can appear and disappear on random times. Therefore, the parameter vector associated with a T-length track, $T \in [0, \infty)$, becomes

$$\vec{x}(T) \in \Big(\mathcal{O}(3) \times R^3 \times R_+ \times \mathcal{A}\Big)^{[0,T]} .$$

As tracks will be discretized to sample times $t_1, t_2, \ldots$ the parameter vector associated with an n-length track $\vec{x}(n)$ is an element of $\vec{x}(n) \in \mathcal{X}(n) \equiv (\mathcal{O}(3) \times R^3 \times R_+ \times \mathcal{A})^n$. Since n is unknown, the full parameter space becomes

$$\mathcal{X} \equiv \bigcup_{n=0}^{\infty} \Big(\mathcal{O}(3) \times R^3 \times R_+ \times \mathcal{A}\Big)^n . \quad (4)$$

The posterior density defined over the full parameter space $\mathcal{X} = \cup_{n=0}^{\infty} \mathcal{X}(n)$ is assumed to be of Gibbs form

$$\pi(\vec{x}(n), n) = \frac{e^{-E(\vec{x}(n),n)}}{Z} . \quad (5)$$

3 The Bayes Posterior

The *ideals* $I \in \mathcal{I}$ are what can be observed by an ideal (with no loss of information) observer. The actual observer, however, may only be able to see the elements with loss of information due to projection, observation noise and/or limited accuracy in the sensor. Denote the operation by which the ideal I appears as some object say $I^{\mathcal{D}}$, by deformations $d : \mathcal{I} \rightarrow \mathcal{I}^{\mathcal{D}}$, $d \in \mathcal{D}$, where $\mathcal{D}$ is the set of deformation mechanism, both random and deterministic.

We take a Bayesian approach to the generation of candidate scenes by defining a posterior probability of the ideal I given the measured data $I^{\mathcal{D}}$ according to

$$\pi(I|I^{\mathcal{D}}) = \pi(I)L(I^{\mathcal{D}}|I) \; ,$$

where $L(I^{\mathcal{D}}|I)$ is the data likelihood and $\pi(I)$ is the prior density on the ideal.

ideal projected onto 2-D with noise superimposed at two different time instants. This is representative of the optical imaging component of the algorithm. The bottom row shows the spatial power spectrum from the narrowband tracker at two instants of time, plotted in the azimuth-elevation plane (bright is low power, dark is high power).

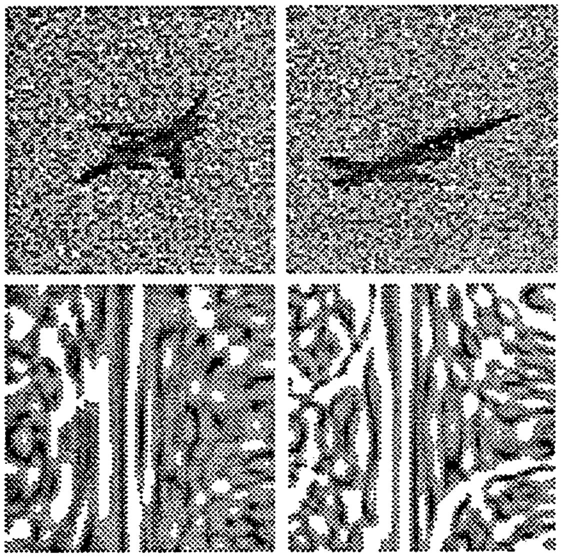


Figure 2: The top row shows the target projected onto the 2-D lattice with additive noise at two different time instants. The bottom row shows the azimuth-elevation signal power profile at two different instants of time as generated from the narrowband tracking data.

The two measurements become $I_1^{\mathcal{P}} \equiv \{\mathbf{y}(t), t \in [0,\infty)\}$, $I_2^{\mathcal{P}} \equiv \{\mathbf{x}(t), t \in [0,\infty)\}$.

4 Jump-Diffusion Random Sampling Algorithm

The most crucial part of the problem still remaining is the derivation of the inference algorithm for generating inferences of high probability. Our approach is to construct a jump-diffusion Markov process following the approach outlined in Grenander and Miller [2] which has the limiting property that it converges in distribution to the Bayes posterior. This implies that time samples of the Markov process will visit the conformations with highest probability most often.

The jump-diffusion Markov process $\{X(t), t \geq 0\}$ samples from the posterior density defined over the full parameter space $\mathcal{X} = \cup_{n=0}^{\infty} \mathcal{X}(n)$ as follows. The jump process, which travels through the infinite number of spaces, carries the inference from space to space with the transition dynamics satisfying the *detailed balance* condition for the jump operator. This coupled with reversibility and connectedness assumptions on the jump statistics allows us to prove irreducibility of the Markov process. In between jump moves the diffusion process searches through the uncountable set of parameters with in each of the subspaces $\mathcal{X}(n)$. As proven in [2, 3, 4] for a large class of Gibb's distributions, with the proper choice of parameters for the Markov process, $\pi(\vec{x}(n), n)$ is the unique stationary density of the jump-diffusion process and the jump-diffusion converges weakly to it. From there it follows that empirical averages generated from realizations converge to their conditional mean expectations.

5 Results

The tracking and recognition algorithms were jointly implemented using a Silicon Graphics workstation for data generation and visualization, and a massively parallel 4096 processor SIMD DECmpp machine for implementing the tracking-recognition random sampling algorithm.

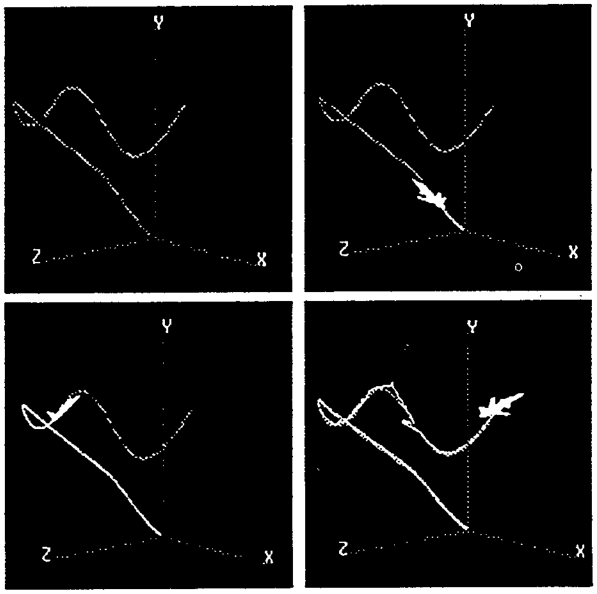


Figure 3: Tracking and recognition using the jump-diffusion algorithm. Top left shows the actual track, the other three panels display the different stages of airplane identification as well as position and orientation estimation.

In the problem stated here the data $I^{\mathcal{D}}$ has multiple components corresponding to the various sensors:

$$I^{\mathcal{D}} \equiv \left(I_1^{\mathcal{D}}, I_2^{\mathcal{D}}, \ldots\right).$$

We only include two sensors for tracking and high-resolution imaging.

3.1 Tracking Priors

The prior on track formation is based on the dynamics of target motion and follows that described in Srivastava et al. [7] in which the force equations governing the motion of targets are utilized to form a prior density on the track parameter space. As an object moves in 3 space it traverses a continuous path consisting of a sequence of translation locations $\vec{p}(t) \in R^3$. For describing their dynamics we shall be interested in expressing the tracks in terms of the velocities of the objects, $\vec{v}(t) \in R^3$ which are related to the positions according to $\vec{p}(t) = \int_{t_0}^{t} \vec{v}(\tau)d\tau + \vec{p}(t_0)$. Following the assumptions in [7] that the target is a rigid body and that the earth's curvature, motion and wind effects are negligible, then the translational motion is given by the Newtonian vector equation

$$\dot{\vec{v}}(t) + A(\vec{\theta}(t))\vec{v}(t) = \vec{f}(t). \tag{6}$$

Here $\vec{\theta}(t) \in [0, 2\pi)^3$ are the Euler's angles representing the orientation of the target with respect to its body frame and

$$A(\vec{\theta}(t)) = \begin{bmatrix} 0 & -q_3(t) & q_2(t) \\ q_3(t) & 0 & -q_1(t) \\ -q_2(t) & q_1(t) & 0 \end{bmatrix},$$

with $\vec{q}(t)$ the rates of change of orientation, which are functions of the Euler's angles.

This linear differential equation is characterized by the time-varying parameter matrix $A(\vec{\theta}(t))$ and force vector $\vec{f}(t)$. The covariance is induced following the approach of Srivastava et al. [7], Amit et al. [8] by assuming the forcing function is a white process with mean $\bar{\vec{f}}(t)$ and a fixed spectral density σ, which then induces a Gaussian process $\vec{v}(t)$ with mean $\bar{\vec{v}}(t)$ and the covariance operator determined by the differential operator of Eqn. (6) according to

$$\bar{\vec{v}}(t) = \int_{t_0}^{t} e^{-\int_{t_1}^{t} A(\tau)d\tau}\,\bar{\vec{f}}(t_1)dt_1 + \bar{\vec{v}}(t_0) \tag{7}$$

$$K_v(t,s) = \sigma \int_{t_0}^{min(t,s)} [e^{-\int_{t_1}^{t} A(\tau)d\tau}][e^{-\int_{t_1}^{s} A(\tau)d\tau}]^T dt_1. \tag{8}$$

Notice, this covariance is parameterized by the sequence of airplane orientations $\vec{\theta}(t), t \in [0, T)$. This connects the tracking and recognition algorithms.

3.2 The Likelihood: Tracking and Imaging Data

There are two sensor types in our problem: a *tracking sensor* and a *high-resolution imaging* sensor.

3.2.1 Tracking

For the tracking we assume a narrowband tracking cross array as in [9, 6, 7] using the standard narrowband signal model developed in [10]. The uniform cross array consists of two uniform linear arrays orthogonal to each other, sensitive to the range, elevation, and azimuth locations of the targets. Since the prior is so naturally defined in velocity coordinates we use the standard cartesian to polar coordinate transformation and the body-centered to inertial-framed [11] transformations to go from velocities to the azimuth, elevation, and range, deterministically. Define this transformation as $\Phi : R^3 \rightarrow [0, 2\pi)^2 \times R_+$.

For the data collected at the P-element sensor array at time t the superposition of the incoming signal and the ambient noise becomes

$$\mathbf{y}(t) = \mathbf{d}(\Phi(\vec{v}(t)))s(t) + \mathbf{n}(t), \tag{9}$$

where $\mathbf{n}(t)$ is a $P \times 1$, 0-mean complex gaussian random vector with identity covariance, $s(t)$ is the signal value and $\mathbf{d}(\Phi(\vec{v}(t)))$ is a regular $P \times 1$ vandermonde direction vector with the angles of arrival parameterized by the velocity vector $\vec{v}(t)$. The *deterministic signal model* is used [9] in which the measurements $\mathbf{y}(t)$ are Gaussian distributed with mean $\mathbf{d}(\Phi(\vec{v}(t)))s(t)$.

3.2.2 Imaging likelihood

While we are currently incorporating models for high-resolution radar imaging as described in [12, 13, 14, 15, 16, 17], all of the results shown are based on optical imaging systems in which the data is assumed to be on a discrete square 64×64 lattice $\mathcal{L}$. The imaging data at time t becomes the set of 256 grey level pixel values $\mathbf{x}(t) \equiv \{x_i, i \in \mathcal{L}\}$. The deformation of the imaging process of the ideal targets is assumed here to be projection with additive noise. For the simulations shown below a Gaussian noise model was used, with the measured data having mean the true ideal 3-D image projected onto the 2-D lattice space.

Shown in Figure 2 are the two kinds of data which the algorithms are based on. The top row shows the

Using a flight simulator on the Silicon Graphics, data was generated for a single airplane flight. From this, tracking data was simulated on the DECmpp assuming a narrowband cross-array of two 32-element uniform linear arrays orthogonal to each other. The track estimation proceeds by births and deaths of track segments at random times through discrete jump moves with a stochastic gradient algorithm run between the jumps for adjusting the orientation and position estimates. In Figure 3 the top left panel shows the actual flight path generated via the flight simulator. The other three panels display the successive stages of the tracking and estimation. The estimated track is superimposed in white on the actual track.

The tracking algorithm utilizes a prior measure on track formation which is parameterized by the orientation motion of the airplane. Using the current estimates of the airplane's position from the tracking, the orientation parameters are simultaneously estimated. Optical imaging of the space around the estimated position is simulated using the Silicon Graphics, as shown in Figure 2. This data set is then transferred to the DECmpp, where the best match is selected from a pre-stored set of 5000 templates sampling the entire object space, all of the possible translations and orientations of the plane.

6 Acknowledgments

This research was supported by ONR N00014-92-J-1418, ARO P-29349-MA-SDI, Rome Laboratory F30602-92-C-004.

References

[1] U. Grenander. Advances in pattern theory: The 1985 Rietz lecture. *The Annals of Statistics*, 17:1–30, 1985.

[2] U. Grenander and M. I. Miller. Jump-diffusion processes for abduction and recognition of biological shapes. *Monograph of the Electronic Signals and Systems Research Laboratory*, 1991.

[3] U. Grenander and M. I. Miller. Representations of knowledge in complex systems. *Journal of the Royal Statistical Society*, in review February 1992.

[4] Y. Amit, U. Grenander, and M.I. Miller. Ergodic properties of jump-diffusion processes. *Annals of Applied Probability*, submitted December 1992.

[5] W. Freiberger and U. Grenander. Computer generated image algebras. *Internation Federation Information Processing*, 68:1397–1404, 1969.

[6] A. Srivastava, M.I. Miller, and U. Grenander. Jump-diffusion processes for object tracking and direction finding. In *Proceedings of the 29th Annual Allerton Conference on Communication, Control and Computing*, pages 563–570, Urbana, Champaign, 1991. University of Illinois.

[7] A. Srivastava, N. Cutaia, M.I. Miller, J. A. O'Sullivan, and D. L. Snyder. Multi-target narrowband direction finding and tracking based on motion dynamics. In *Proceedings of the 30th Annual Allerton Conference on Communication, Control and Computing*, Urbana, Champaign, 1992. University of Illinois.

[8] Y. Amit, U. Grenander, and M. Piccioni. Structural image restoration through deformable templates. *J. American Statistical Association*, 86(414):376–387, June 1991.

[9] M.I. Miller and D. R. Fuhrmann. Maximum likelihood narrow-band direction finding and the EM algorithm. *IEEE Acoust. Speech and Signal Processing*, 38, No.9:560–577, 1990.

[10] R. Schmidt. *A signal subspace approach to multiple emitter location and spectral estimation.* Ph.D. Dissertation of Stanford University, Palo Alto, CA., Nov. 1981.

[11] Bernard Friedland. *Control System Design : An Introduction To State-Space Methods.* McGraw-Hill Book Company, 1986.

[12] D.L. Snyder, J.A. O'Sullivan, and M.I. Miller. The use of maximum-likelihood estimation for forming images of diffuse radar-targets. In *Transactions of SPIE in Advanced Architectures and Algorithms*, San Diego, California, 1987.

[13] D.L. Snyder, J.A. O'Sullivan, and M.I. Miller. The use of maximum-likelihood estimation for forming images of diffuse radar-targets from delay-doppler data. *IEEE Transactions on Information Theory*, pages 536–548, 1989.

[14] J.A. O'Sullivan, P. Moulin, and D.L. Snyder. Cramer-rao bounds for constrained spectrum estimation with application to a problem in radar imaging. In *Proceedings 26th Allerton Conference on Communication, Control, and Computing*, Champaigne, Urbana, October 1988. Urbana, IL.

[15] M.I. Miller, D.R. Fuhrmann, J.A. O'Sullivan, and D.L. Snyder. Maximum-likelihood methods for toeplitz covariance estimation and radar imaging. In Simon Haykin, editor, *Advances in Spectrum Estimation*, pages 145–172. Prentice-Hall, 1990.

[16] P. Moulin, J.A. O'Sullivan, and D.L. Snyder. A method of sieves for multiresolution spectrum estimation and radar imaging. *IEEE Transactions on Information Theory*, 1992.

[17] J.A. O'Sullivan, K. C. Du, R. S. Teichman, M.I. Miller, D.L. Snyder, and V.C. Vannicola. Radar target recognition using shape models. In *Proc. 30th Annual Allerton Conference on Communication, Control, and Computing*, pages 515–523, Urbana, Il., 1992. University of Illinois.

Target Tracking and Recognition Using Jump-Diffusion Processes *

Anuj Srivastava, Robert S. Teichman, Michael I. Miller

Electronic Systems and Signals Research Laboratory
Department of Electrical Engineering
Washington University,
St. Louis, Missouri 63130

1 Introduction

In this paper a new approach for target tracking and recognition is presented. We take a Bayesian approach and define a prior density on the scenes of targets and combine it with likelihoods based on the sensor data to give a posterior measure on the parameter space. The jump-diffusion random sampling algorithm [1, 2, 3] is used to sample from the posterior.

In Section 2 the basic approach for solving the problem is described. The global shape model approach is described in section 3 with the Bayesian posterior measure derived in section 4. Section 5 illustrates the use of jump-diffusion processes for random sampling from the posterior measure over the parameter space. The last section describes the implementation on a parallel processing machine and the results obtained.

2 Recognition Via Deformable Templates

A fundamental task in the representation of complex dynamically changing scenes involving rigid targets is the construction of models that incorporate both the variability of orientation, range, and object number, as well as the precise rigid structure of the objects in a mathematically precise way. The global deformable shape models introduced in Grenander's general pattern theory [4] extended to parametric representations of arbitrary unknown model order [1, 2] are intended to do this. This becomes the basis for our approach.

There are various kinds of variability and uncertainty inherent to data obtained via remote sensing via high resolution and tracking radars. The first and foremost variability is associated with the conformations of the rigid bodies: orientations, scales, and position. To accommodate this type of variability we use global templates which are made flexible via the introduction of basic transformations involving both rigid motions of translation and rotation, as well as non-rigid motions such as scale. As these transformations are parameterized by positions in $\Re^3$ and orientations in $[0, 2\pi)^2 \times [0, \pi]$, they are performed using continuous stochastic gradient search. The *second* kind of variability is associated with the model order, or parametric dimension. In any scene there may be multiple, variable numbers of targets, with tracks of variable lengths and the target number unknown apriori and, therefore, to be estimated itself.

We take a Bayesian approach by defining a prior density on the scenes of targets. The prior coupled to the sensor data likelihood gives the Bayes posterior. The conformation is selected to be consistent with the data in the sense that scenes of high probability under the posterior distribution are selected. Our method for generating candidate conformations is to sample from the posterior. For this a new class of random sampling algorithms is used based on jump-diffusion dynamics, introduced in Grenander and Miller [1, 2, 3] which visit candidate solutions according to the posterior density. The original motivation for introducing jump-diffusions is to accommodate the very different continuous and discrete components of the discovery process. Given a conformation associated with a target type, or group of targets, the problem is to track and identify the orientation, translation and scale parameters accommodating the variability manifest in the viewing of each object type. For this, the parameter space is sampled using Langevin stochastic differential equations in which the state vector continuously winds through the translation-rotation-scale space following gradients of the posterior. The second distinct part of the sampling process supports the recognition associated with choosing the target types. The deduction algorithm goes through multiple stages of hypothesis during which the airplane types are being discovered, and some subset of the scene may be only partially "recognized." This is accommodated by defining the second transformation type which jumps between different object types, where a jump may correspond to the hypothesis of a new object in the scene, or a "change of mind" about an object type. The jump intensities are governed by the posterior density, with the process visiting confor-

---

*Presented at the *ARO's Eleventh Army Conference On Applied Mathematics and Computing*, June 8-11,1993, Carnegie Mellon University, Pittsburgh, PA. This research was supported by ONR N00014-92-J-1418, ARO DAA 03-92-6-0141, Rome Laboratory F30602-92-C-004.

mations of higher probability for longer exponential times, and the diffusion equation governing the dynamics between jumps.

3 Global Templates

We use the global shape models and pattern theoretic approach introduced by Grenander [5, 4]. As the basic building blocks of the hypotheses define the set of generators $\mathcal{G}$, the targets placed at the origin of the *reference coordinates* at a fixed orientation, position, and unit scale.
The fundamental variability in target spaces is accommodated by applying the transformations $T(\vec{\phi}), T(\vec{p}), T(s)$ to the templates $\mathcal{G}$ according to

$$T(\vec{\phi}) \; : \; \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \to \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi_1 & sin\phi_1 \\ 0 & -sin\phi_1 & cos\phi_1 \end{bmatrix} \times \qquad (1)$$

$$\begin{bmatrix} cos\phi_2 & 0 & sin\phi_2 \\ 0 & 1 & 0 \\ sin\phi_2 & 0 & cos\phi_2 \end{bmatrix} \begin{bmatrix} cos\phi_3 & sin\phi_3 & 0 \\ -sin\phi_3 & cos\phi_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$T(\vec{p}) \; : \; \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \to \begin{bmatrix} x_1 + p_1 \\ x_2 + p_2 \\ x_3 + p_3 \end{bmatrix} \qquad (2)$$

$$T(s) \; : \; \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \to \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & s \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} , \qquad (3)$$

where $\vec{\phi}$ is the triple of rotation angles associated with the rotation of the viewing sphere about each object, $\vec{p}$ are the translation parameters in $\Re^3$, and the s is the scale parameter in $\Re_+$. These parameterized transformations operate on the template targets of $\mathcal{G}$ generating the full target space.

Figure 1 shows one of the 3-D ideal targets used for all of the simulations below. The left panel shows a rendering of the target at the origin, the right panel showing the result of applying one of the transformations.

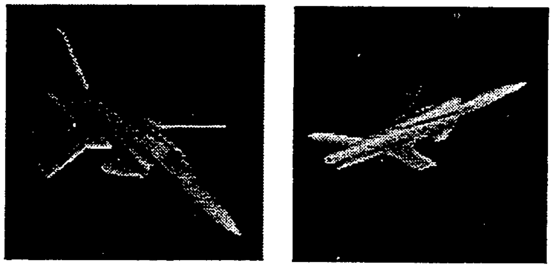


Figure 1: 3-D target at the origin (left panel) and after applying transformations (right panel)

3.1 The Parametric Space

Now the parametric space parameterizing the Bayes posterior becomes the set of parameters specifying the similarity transformations, as well as the airplane type. Define the space containing orientations $\vec{\phi}$ as $\mathcal{M}(3) \equiv [0, 2\pi)^2 \times [0, \pi]$ with 0 and 2π identified. The position vector $\vec{p}$ lies in $\Re^3$ with the scale parameter belonging to $\Re_+$. Then associated with each target or generator $g \in \mathcal{G}$ is a parameter vector $\vec{x} \in \mathcal{M}(3) \times \Re^3 \times \Re_+ \times \mathcal{A}$, where $|\mathcal{A}| = |\mathcal{G}|$ the number of different target types.

A pattern will be constructed from multiple targets with varying track lengths. In [6] we have described the multi-target scenario. Here we focus on single target scenes. We are interested in tracking and recognition in "hostile/non-cooperative" environments in which the objects can appear and disappear on random times $T_1, T + T_1 \in [0, \infty)$ with $T \geq 0$. The parameter vector associated with track becomes

$$\vec{x}_T \in \left(\mathcal{M}(3) \times \Re^3 \times \Re_+ \times \mathcal{A}\right)^{[T_1, T+T_1]} \times \Re_+ .$$

As tracks will be discretized to sample times $t_1, t_2, \ldots$, with the object entering and leaving at $n_1, n + n_1$ respectively, the parameter vector $\vec{x}_n$ associated with an n-length track is an element of $\mathcal{X}_n \equiv \left(\mathcal{M}(3) \times \Re^3 \times \Re_+ \times \mathcal{A}\right)^n \times \mathcal{Z}_+$. Since n is unknown, the full parameter space becomes

$$\mathcal{X} \equiv \bigcup_{n=0}^{\infty} \left(\mathcal{M}(3) \times \Re^3 \times \Re_+ \times \mathcal{A}\right)^n \times \mathcal{Z}_+ . \qquad (4)$$

The posterior density defined over the full parameter space $\mathcal{X}$ is assumed to be of Gibbs form

$$\pi(\vec{x}_n) = \frac{e^{-E(\vec{x}_n)}}{Z} . \qquad (5)$$

In the work presented here only rigid transformations are used with $s = 1$. Also we assume the enterance time $T_1(n_1)$ known apriori.

4 The Bayes Posterior

The *ideals* $I \in \mathcal{I}$ are what can be observed by an ideal (with no loss of information) observer. The actual observer, however, may only be able to see the elements with loss of information due to projection, observation noise and/or limited accuracy in the sensor. Denote the operation by which the ideal I appears as some object say $I^{\mathcal{D}}$, by deformations $d : \mathcal{I} \to \mathcal{I}^{\mathcal{D}}$, $d \in \mathcal{D}$, where $\mathcal{D}$ is the set of deformation mechanisms, both random and deterministic.

We take a Bayesian approach to the generation of candidate scenes by defining a posterior probability of the parameter vector $\vec{x}_n$ representing ideal I given the measured data $I^{\mathcal{D}}$ according to

$$\pi(\vec{x}_n) = \frac{1}{Z} e^{-E(\vec{x}_n)} \propto \frac{1}{Z} e^{-(P(\vec{x}_n) + L(I^{\mathcal{D}} | \vec{x}_n))} ,$$

where $L(I^{\mathcal{D}}|\vec{x}_n)$ is the potential associated with data likelihood and $P(\vec{x}_n)$ is the potential associated with the prior density on the parameter space.

In the problem stated here the data $I^{\mathcal{D}}$ has multiple components corresponding to the various sensors:

$$I^{\mathcal{D}} \equiv \left(I_1^{\mathcal{D}}, I_2^{\mathcal{D}}, \dots \right) .$$

We only include two sensors, one for tracking and one for high-resolution imaging.

4.1 Tracking Priors

The prior on track formation is based on the dynamics of target motion and follows that described in Srivastava et al. [7] in which the force equations governing the motion of targets are utilized to form a prior density on the track parameter space. As an object moves in 3 space it traverses a continuous path consisting of a sequence of translation locations $\vec{p}(t) \in \Re^3$. For describing their dynamics we shall be interested in expressing the tracks in terms of the body frame velocities of the objects, $\vec{v}(t) \in \Re^3$ which are related to the inertial positions [8] according to

$$\vec{p}(t) = \int_{T_1}^{t} \Phi(\tau)\vec{v}(\tau)d\tau + \vec{p}(T_1). \tag{6}$$

where $\Phi(\tau)$ is the product of three rotation matrices in Eqn (2). As in [7] the rigid body analysis with the assumptions that the earth's curvature, motion and wind effects are negligible implies that the translational motion is given by the Newtonian vector equation

$$\dot{\vec{v}}(t) + A(\vec{\phi}(t))\vec{v}(t) = \vec{f}(t). \tag{7}$$

Here $\vec{\phi}(t) \in \mathcal{M}(3)$ are the Euler's angles representing the orientation of the target with respect to the ground based inertial frame and

$$A(\vec{\phi}(t)) = \begin{bmatrix} 0 & -q_3(t) & q_2(t) \\ q_3(t) & 0 & -q_1(t) \\ -q_2(t) & q_1(t) & 0 \end{bmatrix},$$

with $\vec{q}(t)$ the rates of change of orientation, which are functions of the Euler's angles.

This linear differential equation is characterized by the time-varying parameter matrix $A(\vec{\phi}(t))$ and force vector $\vec{f}(t)$. The covariance is induced following the approach of Srivastava et al. [7] and Amit et al. [9] by assuming the forcing function is a white process with mean $\vec{\bar{f}}(t)$ and a fixed spectral density σ, which then induces a Gaussian process $\vec{v}(t)$ with mean $\vec{\bar{v}}(t)$ and covariance operator determined by the differential operator of Eqn. (7) according to

$$\vec{\bar{v}}(t) = \int_{T_1}^{t} e^{-\int_{t_1}^{t} A(\phi(\tau))d\tau} \vec{\bar{f}}(t_1)dt_1 + \vec{\bar{v}}(T_1) \tag{8}$$

$$K_v(t,s) = \sigma \int_{T_1}^{min(t,s)} [e^{-\int_{t_1}^{t} A(\phi(\tau))d\tau}][e^{-\int_{t_1}^{s} A(\phi(\tau))d\tau}]^{\dagger} dt_1. \tag{9}$$

Using Eqn (6) the prior density on the airplane positions can be written as

$$K_p(t,s) = \int_{T_1}^{t} \int_{T_1}^{s} \Phi(\tau_1)K_v(\tau_1,\tau_2)\Phi^{\dagger}(\tau_2)d\tau_1 d\tau_2 . \tag{10}$$

Notice, this covariance is parameterized by the sequence of airplane orientations $\vec{\phi}(t), t \in [T_1, T_2)$. This connects the tracking and recognition algorithms.

4.2 The Likelihood: Tracking and Imaging Data

There are two sensor types in our problem: a *tracking sensor* and a *high-resolution imaging* sensor.

4.2.1 Tracking

For the tracking we assume a narrowband tracking cross array as in [10, 6, 7, 11] using the standard narrowband signal model developed in [12]. The uniform cross array consists of two uniform linear arrays orthogonal to each other, sensitive to the range, elevation, and azimuth locations of the targets related to the inertial positions $\vec{p}(t)$ through regular coordinate transformations.

For the data collected at the P-element sensor array at time t the superposition of the incoming signal and the ambient noise becomes

$$\mathbf{d}_{track}(t) = \mathbf{d}(\vec{p}(t))s(t) + \mathbf{n}(t), \tag{11}$$

where $\mathbf{n}(t)$ is a $P \times 1$, 0-mean complex gaussian random vector with identity covariance, $s(t)$ is the signal value and $\mathbf{d}(\vec{p}(t))$ is a regular $P \times 1$ vandermonde direction vector with the angles of signal arrival parameterized by the inertial postiton $\vec{p}(t)$. The *deterministic signal model* is used as in [10] in which the measurements $\mathbf{y}(t)$ are Gaussian distributed with mean $\mathbf{d}(\vec{p}(t))s(t)$.

4.2.2 Imaging

While we are currently incorporating models for high-resolution radar imaging as described in [13, 14, 15, 16, 17, 18], all of the results shown are based on optical imaging systems in which the data is assumed to be on a discrete square 64×64 lattice $\mathcal{L}$. The imaging data at time t is the set of 4096 grey scale pixel values $\mathbf{d}_{recog}(t) \equiv \{d_i(t), i \in \mathcal{L}\}$. The deformation of the imaging process of the ideal targets is assumed here to be projection with additive noise. For the simulations shown below a Gaussian noise model was used, with the measured data having mean the true ideal 3-D image projected onto the 2-D lattice space.

Shown in Figure 2 are the two kinds of data which the algorithms are based on. The left column shows the ideal projected onto 2-D with additive noise at three different time instants. This is representative of the optical imaging component of the algorithm. The right column shows the spatial power spectrum from the narrowband tracker

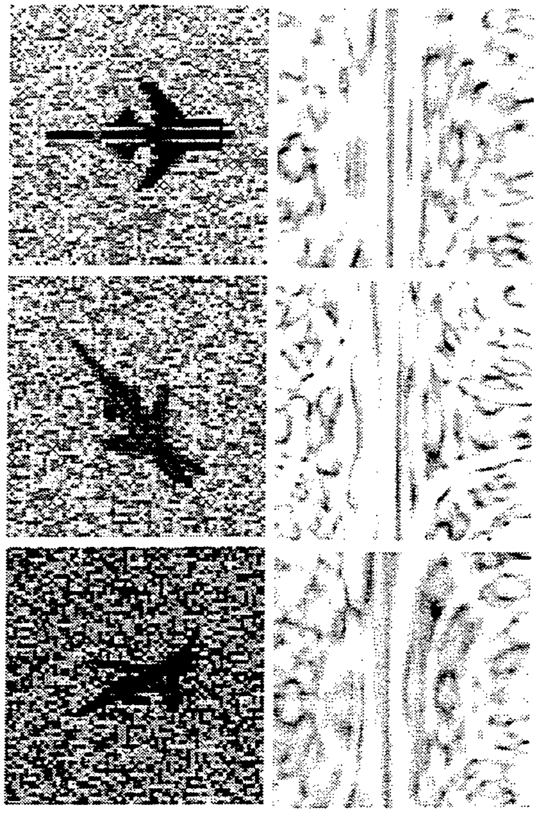


Figure 2: The left column shows the target projected onto the 2-D lattice with additive noise at three different time instants. The right column shows the azimuth-elevation signal power profile at three different instants of time as generated from the narrowband tracking data.

at three instants of time, plotted in the azimuth-elevation plane (bright is low power, dark is high power).

The two measurements become $I_1^{\mathcal{D}} \equiv \{\mathrm{d}_{track}(t), t \in [0,\infty)\}$, $I_2^{\mathcal{D}} \equiv \{\mathrm{d}_{recog}(t), t \in [0,\infty)\}$.

5 Jump-Diffusion Random Sampling Algorithm

The most crucial part of the problem still remaining is the derivation of the inference algorithm for generating inferences of high probability. Our approach is to construct a jump-diffusion Markov process, following the approach outlined in Grenander and Miller [1], which has the limiting property that it converges in distribution to the Bayes posterior. This jump-diffusion Markov process $\{X(t), t \geq 0\}$ samples the posterior density $\pi(\vec{x}_n)$, $\vec{x}_n \in \mathcal{X}$ defined over the full parameter space $\mathcal{X} = \cup_{n=0}^{\infty}\mathcal{X}_n$, i.e. the time samples of the process visit the conformations according to the posterior density. This result is presented in [1, 2, 3] as theorems which follow from two fundamental results. First it is shown that $\pi(\vec{x}_n)$ is a stationary measure of the process $\{X(t), t \geq 0\}$, second the proof that it is the unique stationary measure is presented. The first part is proven by showing that the backward kolmogoroff operator A associated with the Markov process satisfies the condition $\int_{\mathcal{X}} Af(\vec{x}_n)\mu(d\vec{x}_n) = 0$, for $f(\cdot) \in \text{domain}(A)$ and $\mu(d\vec{x}_n) = \pi(\vec{x}_n)m(d\vec{x}_n)$, the distribution corresponding to $\pi(\vec{x}_n)$ with $m(\cdot)$ being the lebesgue measure associated with the parameter space $\mathcal{X}_n$. The generator A has two parts, $A = A^j + A^d$, corresponding to the jump and diffusion terms.

5.1 Jump Process

The jump process travels through the infinite number of subspaces carrying the inference from subspace to subspace. The two kinds of jump moves allowed here are addition or deletion of track segments from the track configuration. These jump moves are performed based on the intensity parameters derived from relative posterior energies of the configurations. For $\vec{x}_n$ being the current configuration and $\vec{y}_m$ a possible candidate, $\vec{x}_n, \vec{y}_m \in \mathcal{X}$, define $q(\vec{x}_n, d\vec{y}_m)$ as the transition intensity, $q(\vec{x}_n)$ as the intensity of jumping out of the space containing $\vec{x}_n$, and $Q(\vec{x}_n, d\vec{y}_m)$ as the probability of transition. These are related according to the relations

$$q(\vec{x}_n) = \int_{\mathcal{T}^1(\vec{x}_n)} q(\vec{x}_n, d\vec{y}_m)$$

and

$$Q(\vec{x}_n, d\vec{y}_m) = \frac{q(\vec{x}_n, d\vec{y}_m)}{q(\vec{x}_n)}.$$

where $\mathcal{T}^1(\vec{x}_n)$ is the set of all configurations reachable in one jump move from $\vec{x}_n$. As shown in [2] there are at least two ways of generating these jump intensities from the posterior density, these being analogues of Gibb's sampling and Metropolis [19] based acceptance-rejection. The implementation presented here uses the latter according to which the jump intensity is defined as

$$q(\vec{x}_n, d\vec{y}_m) = e^{-[L(I^{\mathcal{D}}|\vec{y}_m) - L(I^{\mathcal{D}}|\vec{x}_n)]_+} e^{P(\vec{y}_m)} m(d\vec{y}_m) \quad (12)$$

where $[f(\cdot)]_+$ stands for the positive part of the function. The backward kolmogoroff operator A^j for this jump process is given by

$$A^j f(\vec{x}_n) = -q(\vec{x}_n)f(\vec{x}_n) + \int_{\mathcal{T}^1(\vec{x}_n)} q(\vec{x}_n)Q(\vec{x}_n, d\vec{y}_m)f(\vec{y}_m)$$

and $\int_{\mathcal{X}} A^j f(\vec{x}_n)\mu(d\vec{x}_n) = 0$. This makes $\pi(\vec{x}_n)$ stationary for the jump part of the process.

5.2 Diffusion Process

Between jump transitions the diffusion process searches through the uncountable set of parameters within each of the subspaces $\mathcal{X}_n$. It is a sample path continuous process which essentially performs a randomized gradient descent over the posterior potential $E(\vec{x}_n)$ associated with parameter space $\mathcal{X}_n$ according to Langevin's stochastic differential equation (SDE),

$$dX(t) = \nabla E(X(t))dt + \sqrt{2}dW(t) \tag{13}$$

where $W(t)$ is the standard vector wiener process of dimension of the parameter space $\mathcal{X}_n$. The backward kolmogoroff operator A^d defining the diffusion generated by above SDE is given by

$$A^d f(\vec{x}_n) = \nabla E(\vec{x}_n).\nabla f(\vec{x}_n) + \sum_{i,j} \frac{\partial f^2(\vec{x}_n)}{\partial(\vec{x}_n)_i \partial(\vec{x}_n)_j}$$

and satisfies the condition $\int_{\mathcal{X}} A^d f(\vec{x}_n)\mu(d\vec{x}_n) = 0$. See [1] for the proof.

6 Results

The tracking and recognition algorithms were jointly implemented using a Silicon Graphics workstation for data generation and visualization, and a massively parallel 4096 processor SIMD DECmpp machine for implementing the tracking-recognition random sampling algorithm.

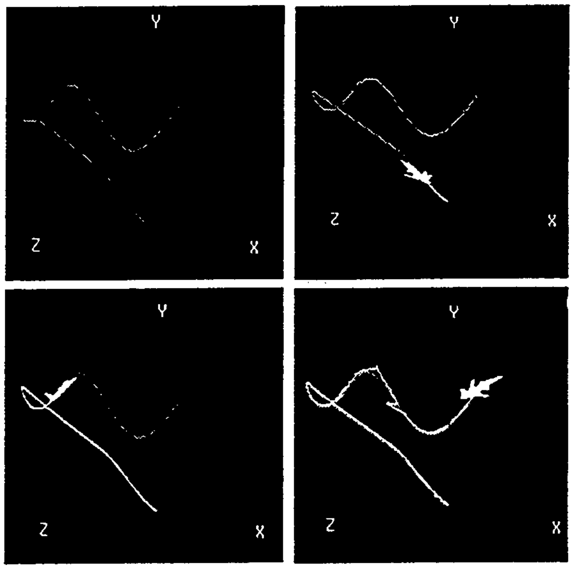


Figure 3: Tracking and recognition using the jump-diffusion algorithm. Top left shows the actual track, the other three panels display the different stages of airplane identification as well as position and orientation estimation.

Using the flight simulator on the Silicon Graphics, a parameterized flight path (Figure 3 top left panel) was generated and stored. From this, tracking data was simulated on the DECmpp assuming a narrowband cross-array of two 32-element uniform linear arrays orthogonal to each other. Optical imaging of the space around the estimated target positions is simulated using the Silicon Graphics to generate data, as shown in Figure 2, which consitutes the imaging data. These two data sets combine to form the posterior density over the parameter space conditioned on the available data up to current time t. For each candidate target type, a set of "snap-shots" sampling efficiently the object space, all the possible orientations of the object, is pre-stored. The position, orientation and target-type parameters are simultaneously estimated using the jump-diffusion algorithm to sample the posterior density. The estimation proceeds by births and deaths of track segments at random times through discrete jump moves. These moves are performed by generating candidate configurations from the prior density, using target dynamics in this case, and selecting using Metropolis acceptance/rejection. A second type of jump move is allowed to sample from the the object space $\mathcal{A}$. A diffusion algorithm is run between the jumps for adjusting the orientation and position estimates following gradients over the posterior energy. The estimation utilizes the prior measure on target positions which is parameterized by the rotational motion of the target. The object recognition is coupled to the target tracking by use of orientation estimates in the prior measure. In Figure 3 the top left panel shows the actual flight path generated via the flight simulator. The other three panels display the successive stages of the tracking and estimation. The estimated track is superimposed in white on the actual track.

References

[1] U. Grenander and M. I. Miller. Jump-diffusion processes for abduction and recognition of biological shapes. *Monograph of the Electronic Signals and Systems Research Laboratory*, 1991.

[2] U. Grenander and M. I. Miller. Representations of knowledge in complex systems. *Journal of the Royal Statistical Society*, to appear October, 1993.

[3] Y. Amit, U. Grenander, and M.I. Miller. Ergodic properties of jump-diffusion processes. *Annals of Applied Probability*, submitted December 1992.

[4] U. Grenander. Advances in pattern theory: The 1985 rietz lecture. *The Annals of Statistics*, 17:1–30, 1985.

[5] W. Freiberger and U. Grenander. Computer generated image algebras. *Internation Federation Information Processing*, 68:1397–1404, 1969.

[6] A. Srivastava, M. I. Miller, and U. Grenander. Jump-diffusion processes for object tracking and direction finding. In *Proc. 29th Annual Allerton Conference on Communication ,Control and Computing*, pages 563–570, Urbana, Il, October 1991. University of Illinois.

[7] A. Srivastava, N. Cutaia, M. I. Miller, J. A. O'Sullivan, and D. L. Snyder. Multi-target narrowband direction finding and tracking using motion dynamics. In *Proc. 30th Annual Allerton Conference on Communication, Control, and Computing*, pages 279–288, Urbana, Il, October 1992. University of Illinois.

[8] Bernard Friedland. *Control System Design : An Introduction To State-Space Methods.* McGraw-Hill Book Company, 1986.

[9] Y. Amit, U. Grenander, and M. Piccioni. Structural image restoration through deformable templates. *J. American Statistical Association*, 1991.

[10] M.I. Miller and D. R. Fuhrmann. Maximum likelihood narrow-band direction finding and the em algorithm. *IEEE Acoust. Speech and Signal Processing*, 38, No.9(38, No.9):560–577, 1990.

[11] M.I. Miller, R. S. Teichman, A. Srivastava, J.A. O'Sullivan, and D. L. Snyder. Jump-diffusion processes for automated tracking-target recognition. In *1993 Conference on Information Sciences and Systems*, Baltimore, Maryland, March 24-26 1993. Johns Hopkins University.

[12] R. Schmidt. *A signal subspace approach to multiple emitter location and spectral estimation.* Ph.D. Dissertation of Stanford University, Palo Alto, CA., Nov. 1981.

[13] D.L. Snyder, J.A. O'Sullivan, and M.I. Miller. The use of maximum-likelihood estimation for forming images of diffuse radar-targets. In *Transactions of SPIE in Advanced Architectures and Algorithms*, San Diego, California, 1987.

[14] D.L. Snyder, J.A. O'Sullivan, and M.I. Miller. The use of maximum-likelihood estimation for forming images of diffuse radar-targets from delay-doppler data. *IEEE Transactions on Information Theory*, pages 536–548, 1989.

[15] J.A. O'Sullivan, P. Moulin, and D.L. Snyder. Cramer-rao bounds for constrained spectrum estimation with application to a problem in radar imaging. In *Proceedings 26th Allerton Conference on Communication, Control, and Computing*, Champaigne, Urbana, October 1988 October 1988. Urbana, IL.

[16] M.I. Miller, D.R. Fuhrmann, J.A. O'Sullivan, and D.L. Snyder. Maximum-likelihood methods for toeplitz covariance estimation and radar imaging. In Simon Haykin, editor, *Advances in Spectrum Estimation.* Prentice-Hall, 1990.

[17] P. Moulin, J.A. O'Sullivan, and D.L. Snyder. A method of sieves for multiresolution spectrum estimation and radar imaging. *to appear in IEEE Transactions on Information Theory*, 1992.

[18] J.A. O'Sullivan, K. C. Du, R. S. Teichman, M.I. Miller, D.L. Snyder, and V.C. Vannicola. Radar target recognition using shape models. In *Proc. 30th Annual Allerton Conference on Communication, Control, and Computing*, pages 515–523, Urbana, Il., 1992. University of Illinois.

[19] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Physical Chemistry*, 21:1087, 1953.

MULTI-TARGET NARROWBAND DIRECTION FINDING AND TRACKING USING MOTION DYNAMICS

Anuj Srivastava and Nicholas Cutaia and
Michael I. Miller and Joseph A. O'Sullivan and Donald L. Snyder

1 Introduction

Multiple target tracking and narrowband direction finding are well known problems in the signal processing literature. In [1] we have described a new family of random sampling algorithms for use in multi-target tracking of dynamically moving targets by passive sensor arrays. We take a Bayesian approach where we define a posterior probability over the configuration space of multiple tracks and use a jump-diffusion algorithm to generate conditional mean estimates from the probability measure. The mathematical foundation for jump-diffusion processes used for model-order estimation problems of this kind is developed in [2, 3].

In this paper we present a new prior on track formation which is based on the dynamics of target motion. The force equations governing the motion of targets are utilized to form a prior density on the multi-track parameter space. This combined with the likelihood provides the posterior measure for the jump-diffusion algorithm.

All authors are with Electronic Signals and Systems Research Laboratory, Department of Electrical Engineering at Washington University, St. Louis , Missouri 63130.

This paper was presented at the 30th Annual Allerton Conference on Communication, Control and Computing, 1992, University of Illinois, Urbana-Champaign, Illinois.

This work was supported by Rome Laboratory contract number F30602-92-C-0004, by the ARO P-29349-MA-SDI and ONR 59922.

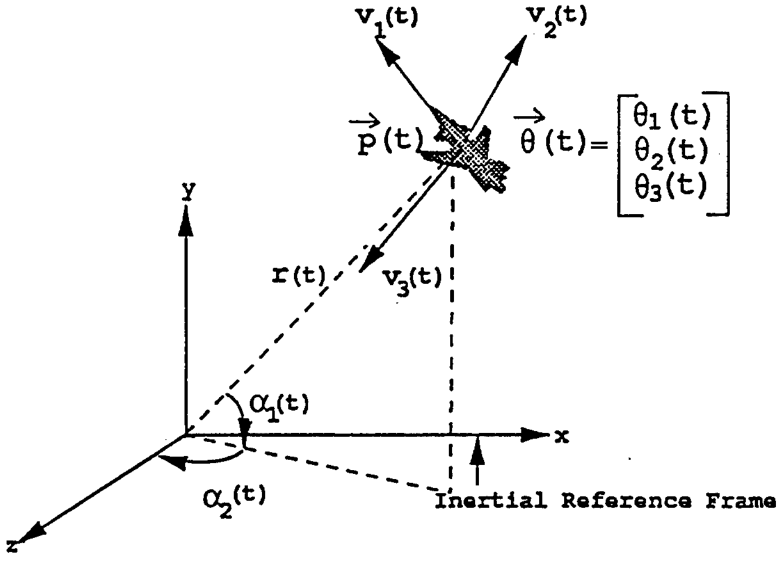


Figure 1: Aircraft motion in 3-D space, $\vec{\theta}(t)$ representing the rotational motion while $\vec{p}(t)$ signifies the translational motion.

Section 2 analyzes the aircraft motion so as to induce the prior on track formation, with section 3 deriving the signal model and the Bayes posterior. Section 4 shows various results on multiple target tracking using the jump-diffusion algorithm.

2 Motion Dynamics Induced Prior

As an object moves in 3 space it traverses a continuous path consisting of a sequence of point locations $\vec{p}(t) = [p_1(t), p_2(t), p_3(t)]^T \in \Re^3$ as shown in Figure (1). For describing their dynamics we shall be interested in expressing the tracks in terms of the velocities of the objects, $\vec{v}(t) = [v_1(t), v_2(t), v_3(t)]^T \in \Re^3$ which are related to the positions according to $\vec{p}(t) = \int_{t_0}^{t} \vec{v}(\tau)d\tau + \vec{p}(t_0)$.

In defining the prior we make the following assumptions: (i) the target is a rigid body, (ii) the earth's curvature is negligible, (iii) the earth's motion is negligible compared to target motion, and (iv) wind effects are negligible. Under these conditions, the translational motion of the target is governed by the force equations based on Newton's second law of motion, given by

$$\begin{aligned}
\dot{v}_1(t) + q_2(t)v_3(t) - q_3(t)v_2(t) &= \frac{1}{m}(f_1(t) - wsin(\theta_1(t))), \\
\dot{v}_2(t) + q_3(t)v_1(t) - q_1(t)v_3(t) &= \frac{1}{m}(f_2(t) + wsin(\theta_2(t))cos(\theta_1(t))), \\
\dot{v}_3(t) + q_1(t)v_2(t) - q_2(t)v_1(t) &= \frac{1}{m}(f_3(t) + wcos(\theta_2(t))cos(\theta_1(t))) .
\end{aligned}$$

Each equation corresponds to the motion along one of the principle axes. In these equations, $f_1(t), f_2(t)$, and $f_3(t)$ are the applied linear forces on the target, w and m are the weight and mass of the target, $\vec{\theta}(t) = (\theta_1(t), \theta_2(t), \theta_3(t)) \in [0, 2\pi)^3$ are the Euler angles representing the orientation of the target with respect to its body frame of reference, and $\vec{q}(t) = [q_1(t), q_2(t), q_3(t)]$ are the rates of change of orientation, which are functions of Euler's angles and their derivatives. In vector form this becomes

$$\dot{\vec{v}}(t) + A(\vec{\theta}(t))\vec{v}(t) = \vec{f}(t), \tag{1}$$

where

$$A(\vec{\theta}(t)) = \begin{bmatrix} 0 & -q_3(t) & q_2(t) \\ q_3(t) & 0 & -q_1(t) \\ -q_2(t) & q_1(t) & 0 \end{bmatrix},$$

and

$$\vec{f}(t) = \frac{1}{m}\begin{bmatrix} f_1(t) - wsin(\theta_1(t)) \\ f_2(t) + wsin(\theta_2(t))cos(\theta_1(t)) \\ f_3(t) + wcos(\theta_2(t))cos(\theta_1(t)) \end{bmatrix}$$

This linear differential equation is characterized by the time-varying parameter matrix $A(\vec{\theta}(t))$ and force vector $\vec{f}(t)$. We utilize the vector equation to determine the covariance of the process $\vec{v}(t)$. To do this, we follow the approach of Amit et al. [4] by assuming the forcing function is a white process with mean $\bar{\vec{f}}(t)$ and a fixed spectral density σ, which then induces a Gaussian process $\vec{v}(t)$ with mean $\bar{\vec{v}}(t)$ and the covariance operator determined by the differential operator of Eqn. (1) as follows. The mean and covariance of the velocity process are given by

$$\begin{aligned}
\bar{\vec{v}}(t) &= \int_{t_0}^{t} e^{-\int_{t_1}^{t} A(\tau)d\tau}\bar{\vec{f}}(t_1)dt_1 + \bar{\vec{v}}(t_0) && (2) \\
K_v(t,s) &= \sigma \int_{t_0}^{min(t,s)} [e^{-\int_{t_1}^{t} A(\tau)d\tau}][e^{-\int_{t_1}^{s} A(\tau)d\tau}]^T dt_1 && (3)
\end{aligned}$$

This covariance is parameterized by the sequence of airplane orientations $\{\vec{\theta}(t)\}_{t_0}^{min(t,s)}$ which is modeled as a continuous Markov process. For all of the implementation we use the Markov Von-Mises prior used in [1] but extended to the sphere.

3 The Posterior

As we assume that the sensor array for narrowband tracking provides a discrete set of sample measurements, we discretize the path and the associated velocities as follows. Defining the point in three-space at which the target resides at the l-th time sample $l\Delta$ as $\vec{p}[l]$, then the velocities are assumed to be piecewise constant given by their values at the sample points $l\Delta, l = 1, 2, \ldots, L$ so that $\vec{p}(l) = \sum_{l=1}^{l} \vec{v}(l)\Delta + \vec{p}(0)$.

Now the discrete prior and likelihood are defined.

3.1 Prior

Using the mean and covariance model from Eqns. (2,3) the prior on the discrete set of velocity vectors is derived straightforwardly. First we discretize the equation of motion according to

$$\vec{v}[n+1] = (I - A[n])\vec{v}[n] + \vec{f}[n], \tag{4}$$

where $A[n]$ implies $A(\vec{\theta}[n])$. Now we can define the covariance of the L-length velocity vector of the m-th track $V^{(m)} = \left(\vec{v}^{(m)}[1], \vec{v}^{(m)}[2], \ldots, \vec{v}^{(m)}[L]\right)^T \in \Re^{3L}$ as matrix $K_{V^{(m)}} = E\{(V^{(m)} - \bar{V}^{(m)})(V^{(m)} - \bar{V}^{(m)})^T\}$, which is an $L \times L$ block matrix with 3 x 3 sub-block elements. As the driving function is assumed white with covariance $E\{(\vec{f}[j] - \bar{\vec{f}}[j])(\vec{f}[k] - \bar{\vec{f}}[k])^T\} = \sigma I \delta[j-k]$, the element $(K_{V^{(m)}})_{j,k}$ with block indices j,k is given by

$$E\{(\vec{v}[j] - \bar{\vec{v}}[j])(\vec{v}[k] - \bar{\vec{v}}[k])^T\} = \sigma \sum_{i=1}^{min(j,k)-1} M[i+1, j-1] M^T[i+1, k-1]\,. \tag{5}$$

where $M[j+1, n]$ is a 3 x 3 matrix given by

$$M[j+1, n] = \prod_{l=j+1}^{n} (I - A[(n+j+1) - l]),$$

and $M[n+1,n]$ is defined to be the identity matrix. The mean velocity vector at any time instant $n+1$ becomes

$$\bar{\vec{v}}[n+1] = \sum_{j=1}^{n} M[j+1,n]\vec{\bar{f}}[j] + M[1,n]\bar{\vec{v}}[1].$$

Given the mean velocities and the covariance matrix, the prior on the m-th track $(V^{(m)}, \Theta^{(m)})$ can be written as

$$\pi(V^{(m)}, \Theta^{(m)}) = (2\pi)^{\frac{-3L}{2}} det^{\frac{-1}{2}} \left[K_{V^{(m)}}\right] e^{-\frac{1}{2}\{(V^{(m)}-\bar{V}^{(m)})^T K_{V^{(m)}}^{-1} (V^{(m)}-\bar{V}^{(m)})+E(\Theta^{(m)}\}}, \tag{6}$$

where $E(\Theta^{(m)})$ represents the prior energy on the rotational motion of the m-th target described by its set of Euler's angles..

3.2 Signal Model & Data Likelihood

To describe the multi-track, multi- sensor scenario we use the standard narrowband signal model developed in [5] and used for tracking in [1], [6]. The targets are assumed observed using a narrowband sensor array, in the experiments shown here a uniform cross array consisting of two uniform linear arrays orthogonal to each other lying in the inertial frame of reference Figure (1). This array is sensitive to the range, elevation, and azimuth locations of the targets $\{\vec{\alpha}(t) : r(t), \alpha_1(t), \alpha_2(t)\}$, respectively. These angles are measured in the inertial frame of reference while the previously defined velocities on which the prior is based are described in the body centered frame. The variables in the two reference frames are related by known transformations [7]. Notice, using the standard cartesian to polar coordinate transformation and the above mentioned transformations, the azimuth, elevation, and range parameters are deterministically related to the velocities. Define a transformation Φ from velocities in body centered frame to the point locations in the inertial frame as

$$\Phi : \{\vec{v}(l)\}_{l=1}^{L} \in \Re^{3L} \longmapsto \{\vec{\alpha}(\tau)\}_{l=1}^{L} \in [0, 2\pi)^{2L} \times \Re_{+}^{L}$$

The transformation Φ allows us to write the likelihood of the sensor data in terms of the velocity vectors.

For the m-th source the data collected at the P-element sensor array at discrete time l is the superposition of the incoming signal and the ambient noise, and is given by

$$\mathrm{y}_m[l] = \mathrm{d}(\Phi(\vec{v}^{(m)}[l]))s_m[l] + \mathrm{n}_m[l], \tag{7}$$

where n[l] is a $P \times 1$, 0-mean complex gaussian random vector with identity covariance, $s_m[k]$ is the signal value and $\mathrm{d}(\Phi(\vec{v}^{(m)}[l]))$ is a regular $P \times 1$ Vandermonde direction vector with the angles of arrival parameterized by the velocity vector $\vec{v}^{(m)}[l]$. In the multiple target case, the received signal is a superposition of the signals from each target. Assuming there are M targets present, the received data in matrix form becomes

$$\mathrm{Y}[l] = \mathrm{D}_M(\Phi(\vec{v}^{(1)}(l)), .., \Phi(\vec{v}^{(M)}(l)))\mathrm{s}_M[l] + \mathrm{n}[l] \tag{8}$$

where $\mathrm{D}_M[l]$ is the $P \times M$ matrix $[\mathrm{d}(\Phi(\vec{v}^{(1)}[l]))\ \mathrm{d}(\Phi(\vec{v}^{(2)}[l]))\ \ldots\ \mathrm{d}(\Phi(\vec{v}^{(M)}[l]))]$, and $\mathrm{s}_M[l]$ is the $M \times 1$ vector of signal amplitudes at time l.

The vector of signal values $\mathrm{s}_M[l]$ is modeled as a deterministic quantity, with the likelihood for an M-track scene becoming

$$L(data|V^{(1)}, \ldots, V^{(M)}) \propto \prod_{l=1}^{L} e^{-|\mathrm{Y}[l]-\mathrm{D}_M(\Phi(\vec{v}^{(1)}(l)),..,\Phi(\vec{v}^{(M)}(l)))s_M[l]|^2}. \tag{9}$$

3.3 Parameter Space and Posterior

To formulate the Bayes posterior first we define the parameter space on which it has support, and then combine the prior from track dynamics with the likelihood.

Define a track-configuration as the set of parameters which completely describe the scene at any time. Suppose in an observation period the motion of the *m-th* source is sampled L times so its track is completely defined by a sequence of L velocity vectors according to, $V^{(m)} = (\vec{v}^{(m)}[1], \ldots, \vec{v}^{(m)}[L])^T \in \Re^{3L}$. For a multiple track scene with M targets, M being unknown, the configuration is the set of M tracks $(V^{(1)}, \ldots, V^{(M)}) \in \Re^{3ML}$, with the complete configuration or parameter space being $\bigcup_{M=0}^{\infty} \Re^{3ML}$. The parameter space is a countable, infinite union of subspaces each having an associated value of M, highlighting an important issue in this estimation problem. The model order, i.e. the number M of targets present in the observation space, is not known beforehand. Each scene of M targets has a variable number 3ML of parameters, with M itself a parameter to be estimated by the algorithm.

Since the prior on motion dynamics involves the trajectories of the Euler angles $\vec{\theta}$, a bigger parameter space is required to account for both the rotational and translational motions . The parameter set for M-tracks becomes $((V^{(1)}, \Theta^{(1)}), \ldots, (V^{(M)}, \Theta^{(M)})) \in \Re^{3ML} \times [0, 2\pi)^{3ML}$, with the complete parameter space being $\bigcup_{M=0}^{\infty} \Re^{3ML} \times [0, 2\pi)^{3ML}$. Now we define the posterior measure over this complete configuration space.

For the multi-target case, the target motion is assumed to be independent across targets, giving the prior

$$\pi((V^{(1)}, \Theta^{(1)}), (V^{(2)}, \Theta^{(2)}), ..., (V^{(M)}, \Theta^{(M)}, M) = \prod_{m=1}^{M} \pi(V^{(m)}, \Theta^{(m)}) \ \pi(M) \ , \tag{10}$$

with $\pi(M)$ the prior on the number of tracks and $\sum_{M=0}^{\infty} \pi(M) = 1$.

Combining the likelihood and prior terms the posterior is of Gibbs form $\pi = \frac{e^{-U}}{Z}$ with potential

$$\begin{aligned} U(V^{(1)}, \Theta^{(1)}), (V^{(2)}, \Theta^{(2)}), \quad \ldots \quad & , (V^{(M)}, \Theta^{(M)}, M) \\ = \quad & \sum_{l=1}^{L} (-|\mathrm{Y}[l] - \mathrm{D}_M(\Phi(\vec{v}^{(1)}[l], ..\Phi(\vec{v}^{(M)}[l])))s_M[l]|^2) \\ - \quad & \sum_{m=1}^{M} [(V^{(m)} - \bar{V}^{(m)})^T K_{V^{(m)}}^{-1} (V^{(m)} - \bar{V}^{(m)})] + \sum_{m=1}^{M} E(\Theta^{(m)}) \\ + \quad & log(\pi(M)). \end{aligned}$$

4 Simulation Results

The estimation is performed by sampling the multi-track parameter space $\bigcup_{M=0}^{\infty} \Re^{3ML} \times [0, 2\pi)^{3ML}$ using jump-diffusion processes as described in [1, 2, 3]. We construct a Markov process which visits the candidate solutions according to the posterior Gibb's density $\pi = \frac{e^{-U}}{Z}$. In this algorithm, the jump process carries the estimate from subspace to subspace of various track numbers, with the diffusion following stochastic gradient descent within the subspaces. It has been proven that the jump-diffusion produces samples from the posterior, so that the empirical averages converge to their conditional mean expectations under the posterior.

Figure (2) demonstrates the use of jump-diffusion based sampling algorithm for multiple target tracking in the plane where the tracks are com-

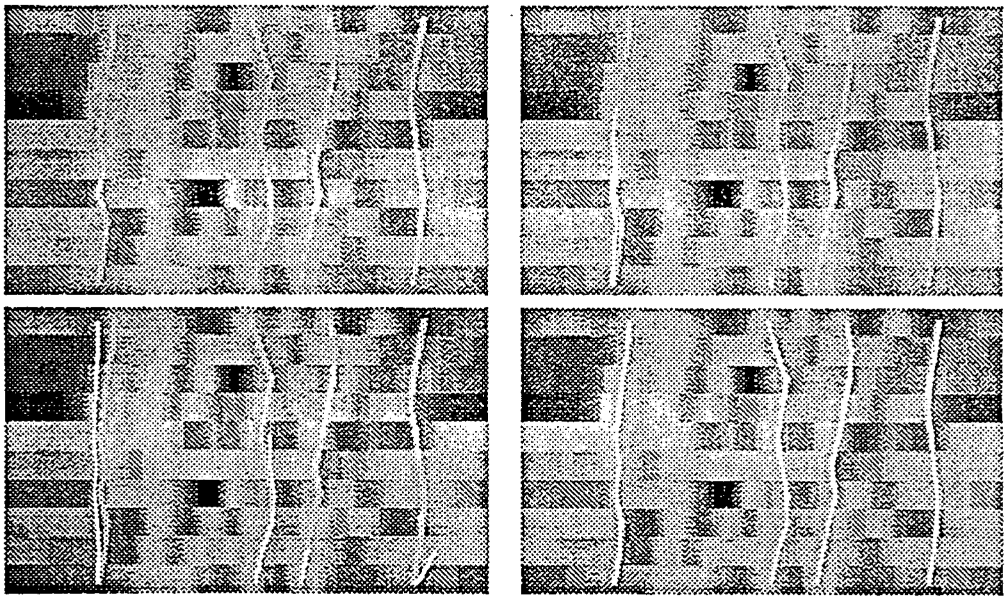


Figure 2: Illustrating jump-diffusion based sampling for 2D problem estimating only one angular location. In the background is the spatial power spectrum with brightness representing energy at that angular location. The top left panels shows the four tracks, top right and bottom left show intermediate stages of estimation with the final result in the bottom right.

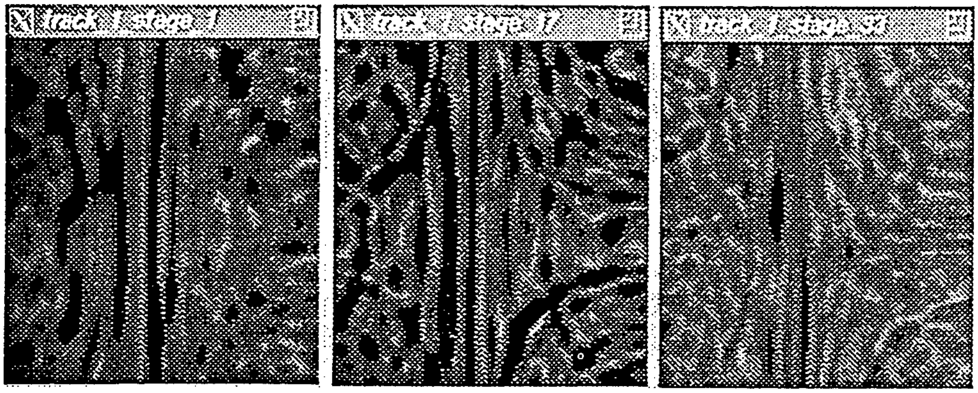


Figure 3: The three panels display the spatial power spectrum at three locations along the track in 3 space with brightness a function of energy.

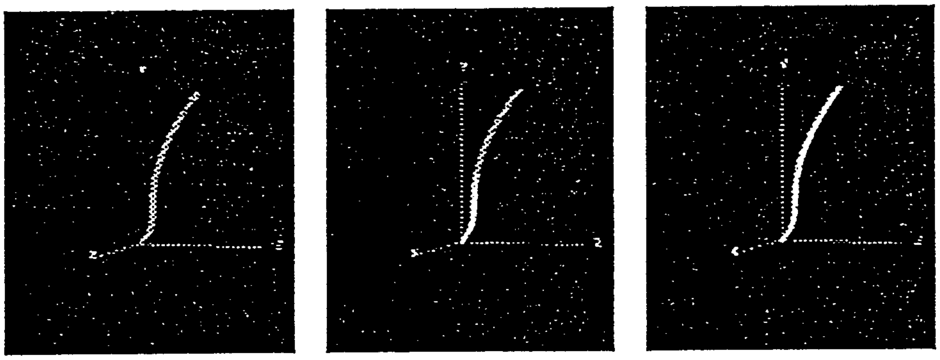


Figure 4: Estimation of single track configuration in 3D space. The left panel shows the true track in gray. The middle and the right panels show intermediate and final track estimates with the estimated track shown overlying in white.

pletely specified via a single elevation angle. The prior used is the Von-Mises prior (see [1] for details of implementation etc). Further the targets are observed by a uniform linear array, with the algorithm forced to estimate the number of targets as well as the paths themselves. The four panels show various stages of the algorithm. The actual tracks are shown in top left while top right and bottom panels show the intermediate and the final results with the estimates drawn over the actual tracks. Shown in the background is the spatial power spectrum of the data generated using *minimum variance distortionless response* (MVDR) beamforming [8] with the brighter spots signifying higher energy locations.

For 3D problem, the algorithm was run on a simulated data set with track length L = 64 and a cross-array having two 32 elements subarrays each with half wavelength spacing. Shown in Figure (3) are a series of spatial power spectra of data at three different time indices along the track. Figure (4) shows the estimation of a complete track from this noisy data set. the left panel shows the actual track in gray in the inertial frame of reference with the middle and right panels showing the estimated track drawn over the actual track at the successive stages of the algorithm.

References

[1] A. Srivastava, M. I. Miller, and U. Grenander. Jump-diffusion processes for object tracking and direction finding. In *Proc. 29th Annual Allerton Conference on Communication ,Control and Computing*, pages 563–570, Urbana, Il, October 1991. University of Illinois.

[2] U. Grenander and M. I. Miller. Jump-diffusion processes for abduction and recognition of biological shapes. *Monograph of the Electronic Signals and Systems Research Laboratory*, 1991.

[3] U. Grenander and M. I. Miller. Representations of knowledge in complex systems. *Journal of the Royal Statistical Society*, in review.

[4] Y. Amit, U. Grenander, and M. Piccioni. Structural image restoration through deformable templates. *J. American Statistical Association*, 1991.

[5] R. Schmidt. *A signal subspace approach to multiple emitter location and spectral estimation.* Ph.D. Dissertation of Stanford University, Palo Alto, CA., Nov. 1981.

[6] M.I. Miller and D. R. Fuhrmann. Maximum likelihood narrow-band direction finding and the em algorithm. *IEEE Acoust. Speech and Signal Processing*, 38, No.9(38, No.9):560–577, 1990.

[7] Bernard Friedland. *Control System Design : An Introduction To State-Space Methods.* McGraw-Hill Book Company, 1986.

[8] N. Owsley. Sonar array processing. In Simon Haykin, editor, *Array Signal Processing.* Prentice-Hall, New Jersey, 1985.

Lie Group Parameterization for Dynamics Based Prior in ATR [*]

Anuj Srivastava [†] Michael I. Miller[†] Ulf Grenander [‡]

Recently the authors have presented a new random sampling algorithm for automated tracking and recognition of multiple targets [1], based on the ideas presented in [2]. This method involves dynamics based tracking in a general scenario, featuring non-linear data equations and having unknown (and variable) number of targets to track/recognize. Taking the Bayesian approach, a posterior measure is constructed on the parameter space $\mathcal{X} = \bigcup_{M=0}^{\infty} \mathcal{X}_M$, where $\mathcal{X}_M$ is the parameter space associated with M targets. Then, a jump-diffusion algorithm is used to sample from this posterior for empirically generating the conditional expectations under this posterior to solve the minimum mean square error (M.M.S.E) problem. Earlier, the estimation problem was solved with the target paths parameterized by the sequence of positions $p \in \Re^3$ and the Euler orientations (pitch, roll and yaw) $\phi \in [0, 2\pi]^3, 0, 2\pi$ identified.

1 Parameterization

Now we present a more natural parameterization, of track scenes, which simplifies the derivation of the dynamics based prior on the parameter space. Here, the target orientation is represented by a 3×3 matrix based on the Euler angles according to

$$O(t) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi_1(t) & sin\phi_1(t) \\ 0 & -sin\phi_1(t) & cos\phi_1(t) \end{bmatrix} \begin{bmatrix} cos\phi_2(t) & 0 & -sin\phi_2(t) \\ 0 & 1 & 0 \\ sin\phi_2(t) & 0 & cos\phi_2(t) \end{bmatrix} \begin{bmatrix} cos\phi_3(t) & sin\phi_3(t) & 0 \\ -sin\phi_3(t) & cos\phi_3(t) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Notice, $O(t)$ belongs to a group of 3×3 orthogonal matrices with determinant 1, called the special orthogonal group $SO(3)$, known to be a Lie group under matrix multiplication. It can be shown that $O(t + \epsilon) = e^{-\epsilon A(q(t))} O(t)$, where $A(q(t))$ is the skew-symmetric matrix consisting of elements of $q(t) = [q_1(t)\ q_2(t)\ q_3(t)]$, the vector of target's rotational velocities in its body-frame and $A(q(\tau)) = A(q(t)), \tau \in [t, t+\epsilon)$. Assume that a target is observed in the interval $[t_0, T]$ then the target path can be described by $\{p(t), O(t)\}^{[t_0,T]} \in (\Re^3 \times SO(3))^{[t_0,T]}$. Observe that the orientation process can be parameterized by the angular velocities of the airplane, with the representation further simplified to $\{v(t), q(t)\}^{[t_0,T]} \in \Re^{6[t_0,T]}$, given $p(t_0), O(t_0)$. We choose to pose the estimation problem in terms of these velocities as it is natural to describe the airplane dynamics via the linear and angular velocities using the Newtonian equations of airplane motion as described below.

2 Bayesian Posterior

The posterior measure is composed of two terms: a data driven likelihood component and a prior knowledge component. The airplane motion (under simplifying assumptions [3]) is described by the stochastic differential

---

[*] For correspondence: anuj@hyperion, phone: (314)-935-7552, fax: (314)-9354842. This work was supported by the ARO DAAL03-92-G-0141, ONR N00014-88-K-289, and Rome Laboratory F30602-92-C-0004 to Michael I. Miller and ARO/MIT DAALO3-92-G-0115, ONR NOOO14-91-J-1021, ARL MDA972-93-1-0012, and NSF DMS-9217655 to Ulf Grenander.

[†] Department of Electrical Engineering, Electronic Signals and Systems Research Laboratory, Washington University, St. Louis, MO 63130.

[‡] Division of Applied Mathematics, Brown University, Providence, RI 02920.

equations,

$$\begin{aligned}
\dot{v}_1(t) - q_3(t)v_2(t) + q_2(t)v_3(t) &= f_1(t) \ , \\
\dot{v}_2(t) + q_3(t)v_1(t) - q_1(t)v_3(t) &= f_2(t) \ , \\
\dot{v}_3(t) - q_2(t)v_1(t) + q_1(t)v_2(t) &= f_3(t) \ , \\
I_1\dot{q}_1(t) - (I_2 - I_3)q_2(t)q_3(t) &= \Gamma_1(t) \ , \\
I_2\dot{q}_2(t) - (I_3 - I_1)q_1(t)q_3(t) &= \Gamma_2(t) \ , \\
I_3\dot{q}_3(t) - (I_1 - I_2)q_2(t)q_1(t) &= \Gamma_3(t) \ .
\end{aligned} \tag{1}$$

The first three equations describe the airplane's translational motion, while the next three describe its rotational motion. These equations of motion are used to induce the prior on track formation. Let the state vector be defined as $x(t) = [v(t)\ q(t)]$ and $f(t)$ be the vector of forcing functions (on rhs of the above equations). Let the continuous target motion be sampled at discrete observation times $0, 1, .., n-1$ and define a transformation $L : \Re^{6n} \rightarrow \Re^{6n}$ such that $L(x(0), .., x(n)) = \{f(0), f(1), .., f(n)\}$ is the compact form of the discretized version of the above differential equations. With the forces driving these equations modeled as i.i.d random variables with the density given by $F_1(\cdot)$, the density on x (by change of variables) can be written as $p(x(0), .., x(n)) = J(x(0), .., x(n)) \prod_{i=1}^{n} F_1(L^i(x(0), .., x(n)))$, where $J(\cdot)$, L^i are the Jacobian and the i^{th} component of L. If the derivatives are approximated by forward differences then the Jacobian reduces to one and can be dropped from the expression.

There are two kinds of observation radars assumed, first, a low resolution radar which responds to the target positions assuming point targets used to generate the tracking data, $y_1(t)$. Secondly, for target identification and orientation estimation a high resolution radar is assumed generating the recognition data, $y_2(t)$. Let the data $y(t) = [y_1(t)\ y_2(t)]$ obtained from the radars be modeled as $y(t) = G(x(t)) + n(t)$ where $G(\cdot)$ is possibly a highly non-linear deformation describing radar operation and $n(\cdot)$ is the additive i.i.d noise. If the density for noise samples is given by $F_2(\cdot)$ then the observed data likelihood can be written as $p(y(0), .., y(n)|x(0), .., x(n)) \propto \prod_{i=0}^{n} F_2(y(i) - G(x(i)))$.

Combing the prior measure and the data likelihood we obtain the posterior density on the parameter space given by, $p(x(0), .., x(n)|y(0), .., y(n)) \propto \prod_{i=0}^{n} F_2(y(i) - G(x(i)))F_1(L^i(x(0), .., x(n)))$.

3 Random Sampling Algorithm

The estimation is performed by sampling the multi-track parameter space using a jump-diffusion processes as described in [1]. We construct a Markov process which visits the candidate solutions according to the posterior Gibb's density. In this algorithm, the jump process carries the estimate from subspace to subspace of various track numbers, with the diffusion following stochastic gradient descent within the subspaces. It has been proven that the jump-diffusion generates samples from the posterior, so that the empirical averages converge to their conditional mean expectations under the posterior ([2]).

References

[1] M. I. Miller, A. Srivastava, and U. Grenander. Conditional-expectation estimation via jump-diffusion processes in multiple target tracking/recognition. *IEEE Transactions on Acoustics,Speech and Signal Processing*, submitted for review, April 1994.

[2] U. Grenander and M. I. Miller. Representations of knowledge in complex systems. *Journal of the Royal Statistical Society*, 56(3), 1994.

[3] Bernard Friedland. *Control System Design : An Introduction To State-Space Methods.* McGraw-Hill Book Company, 1986.

Proceedings of the 33rd
Conference on Decision and Control
Lake Buena Vista, FL - December 1994

AUTOMATIC TARGET RECOGNITION USING KINEMATIC PRIORS

Nicholas J. Cutaia Joseph A. O'Sullivan

Electronic Systems and Signals Research Laboratory
Department of Electrical Engineering
Washington University
St. Louis, MO 63130, USA

ABSTRACT

Traditional automatic target recognition (ATR) systems discriminate based upon target size, target shape, or both. In this paper, an ATR algorithm is proposed that exploits aircraft-class specific kinematics to assess the tracked target's likelihood. Prior information on kinematics includes the physical parameters of the aircraft, allowable input forces to a pilot, and pilot behavior in the aircraft. It is shown that the computation of the likelihood of observed events is intractable. A suboptimal approximation to the likelihood can be computed using a hypothesis reduction method based on the Generalized Pseudo-Bayesian (GPB) class of algorithms. A bound on the L_1 distance of a suboptimal approximate density from the true density is derived.

Keywords: generalized pseudo-Bayesian algorithms, interacting multiple model algorithm, automatic target recognition, tracking, jump-linear systems.

1. Introduction

Historically, techniques designed to solve the ATR problem have been based on the hypothesis that features of objects from different classes lie in easily separated regions of a multidimensional feature set. Most of the features used by researchers are geometric, topological, and/or spectral. For a relatively complete coverage of the topic, see [1, 2]. Until recently [2, 3], aircraft kinematics have not been used as a feature. Libby uses kinematic information to eliminate unlikely aircraft orientation sequences in computing the joint density of observations. To our knowledge, there has been little other consideration of the kinematic features as discriminators in the ATR problem.

In this paper, we quantify prior kinematic information and use it to assess the likelihood of a tracked target. Aircraft have unique physical attributes (mass, moments of inertia, control surface dimensions) that characterize their angular and translational motion due to applied input forces whose knowledge can aid in the identification problem. We propose that identification can be enhanced further by constraining allowable input forces to an aircraft-class specific discrete set. Transitions from one input state to another input state in this set are governed by a Markov transition matrix thus portraying the pilot's behavior unique to the aircraft.

The prior on the input force induces exponentially increasing complexity in the computation of the target likelihood. A suboptimal likelihood can be calculated using the well known Generalized Pseudo-Bayesian (GPB) algorithm [4] or the Interacting Multiple Model (IMM) Algorithm [6]. These algorithms have been used extensively to solve the maneuvering target tracking problem [4, 7, 8], but have not been applied to the target recognition problem.

Performance of the likelihood calculator is directly related to the performance of the GPB algorithms. Performance evaluation of the GPB class of algorithms is well known to be a difficult task. In the past, performance evaluation relied upon Monte Carlo simulation. Non-simulation attempts include global averaging approaches and error bounding techniques. Most recently, Li and Bar-Shalom [9] designed an off-line recursion to predict the average performance of the IMM algorithm which boasts similar performance to the GPB2 algorithm. These methods generally considered the mean square error of the state estimate. In this paper, the performance of the suboptimal estimator is bounded by the L_1 difference between the actual output density and the approximate output density. To our knowledge, bounding of a distance measure from the suboptimal approximation of the density to the true density has not been considered in the literature.

2. Problem Formulation

When an aircraft can be assumed to be a rigid body moving in space, its motion can be considered to have six degrees of freedom. By applying Newton's Second Law to the rigid body, the aircraft equations of motion can be established in terms of translational and angular accelerations which occur as a consequence of forces and moments being applied to the aircraft. These equations of motion are nonlinear, but can be linearized about an operating condition. When finding the minimum mean square error state estimate, the extended Kalman Filter can be used and the equations are continuously linearized about the most recent state estimate. Sensor data are discrete time measurements thus motivating a discrete time approximation of the linearized equations of motion. Thus, we consider the aircraft kinematic model as described by the linear discrete time system

$$x_{k+1} = A_i x_k + u_k + w_k \qquad (1)$$

$$y_k = C x_k + v_k, \qquad (2)$$

where the state x is a n dimensional vector, and the observation vector y_k is m dimensional. It is assumed that x_0 is Gaussian with mean 0 and covariance matrix P_0. The

0-7803-1968-0/94$4.00©1994 IEEE

system matrix, A_i contains coefficients which are conditioned on aircraft type $i \in 1, 2, \ldots, N$, where N is number of possible target types. Each process noise vector, w_k is a zero mean Gaussian random vector with covariance Q, and each measurement noise vector, v_k is a zero mean Gaussian random vector with covariance R. Process noise and measurement noise are assumed uncorrelated. The input states are contained in the set Γ_i which is again specified by the aircraft type i. The number of possible input states available to the pilot of aircraft i is $M_i =| \Gamma_i |$. Transitions in this set are determined by the Markov transition matrix Λ_i.

Using the classical Bayesian approach, a target recognition system is designed that estimates the *a posteriori* probability $P(i \mid Y(k))$, where $Y(k)$ is the vector of observation vectors up to and including y_k. This can be rewritten using Bayes Theorem as

$$P(i \mid Y(k)) = \frac{p(Y(k) \mid i)P(i)}{p(Y(k))}. \tag{3}$$

The prior probability on aircraft type, $P(i)$, is engagement dependent, but for now can be considered uniform across target type. The unconditional joint probability in the denominator is common to all target types and can be neglected during a maximization. Thus, the target-conditioned joint probability of the observations is of interest. It will be shown that this joint probability is a M_i^k order gaussian mixture and its ease of computation is determined by the time step k. The primary focus of this paper is directed toward the use of the kinematic priors in determining the target-conditioned joint density of the observations, a suboptimal approximation to this joint probability density, and the quality of the suboptimal approximation and its effect on $P(i \mid Y(k))$.

For the remainder of the paper, conditioning on aircraft-class i will be neglected for notational convenience.

3. Joint Density on Observations

In this section, the joint density on the observations is derived. The target-conditioned joint probability of the observations can be rewritten as

$$p(Y(k)) = \sum_{U(k-1) \in \Omega^{k-1}} p(Y(k) \mid U(k-1))P(U(k-1)), \tag{4}$$

where $U(k-1)$ denotes the sequence of input state vectors up to and including u_{k-1}, and Ω^{k-1} is the set containing all M^k possible input state sequences.

It is relatively straightforward to compute $P(U(k-1))$. Denote the possible input state vectors as

$$\gamma_j, \quad j = 1, 2, \ldots, M, \tag{5}$$

and consider the Markov transition matrix

$$\Lambda = \begin{bmatrix} \lambda_{11} & \lambda_{12} & \ldots & \lambda_{1M} \\ \lambda_{21} & \lambda_{22} & \ldots & \lambda_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{M1} & \lambda_{M2} & \ldots & \lambda_{MM} \end{bmatrix},$$

where the transition probabilities λ_{ij} are defined as

$$\lambda_{ij} = P(u_k = \gamma_j \mid u_{k-1} = \gamma_i). \tag{6}$$

The sequence probability $P(U(k-1))$ can be written in compact form as

$$P(U(k-1)) = P(u_0) \prod_{n=1}^{k-1} \prod_{j=1}^{M} \prod_{l=1}^{M} \lambda_{jl}^{\delta_{u_{n-1},\gamma_j}\delta_{u_n,\gamma_l}} \tag{7}$$

where δ signifies the Kronecker delta function and $P(u_0)$ is assumed known.

The system equation can be written

$$x_k = A^k x_0 + \sum_{i=1}^{k} A^{k-i}(u_{i-1} + v_{i-1}). \tag{8}$$

It is obvious that $p(y_k \mid U(k-1))$ is gaussian. To compute the unconditional joint density on the observations, the joint probability of the observations conditioned on the input state sequence is needed. This probability can be written

$$p(Y(k) \mid U(k-1)) = p(y_1 \mid u_0) \prod_{j=2}^{k} p(y_j \mid U(j-1)). \tag{9}$$

The product of the gaussian densities $p(y_j \mid U(j-1))$ remains gaussian so the joint probability of the observations (4) is a mixture of M^k gaussians.

As an alternative expression to (4), the joint probability can be expressed as

$$p(Y(k)) = p(y_1) \prod_{j=2}^{k} p(y_j \mid Y(j-1)). \tag{10}$$

This can be rewritten as

$$\begin{aligned} p(Y(k)) \quad = \quad & p(y_1) \prod_{j=2}^{k} \sum_{U(j-1) \in \Omega^{j-1}} p(y_j \mid U(j-1)) \cdot \\ & P(U(j-1) \mid Y(j-1)). \end{aligned} \tag{11}$$

Again, at time step k, the joint probability of the observations involves the computation of an order M^k gaussian mixture. In this form, the density $p(y_j \mid U(j-1))$ as well as the posterior probabilities of the input state sequences can be approximated recursively using the GPB class of algorithms.

4. Approximation of the Mixture Density

The calculation of the joint probability of the observations becomes intractable as the time step k becomes large. To keep the problem tractable, a strategy to consider only the N gaussians of the M^k gaussian mixture that have the largest contribution to the joint probability of the observations is necessary. The challenge incurred is twofold. First, a method for eliminating terms of the mixture that are considered insignificant must be defined. Second, the impact of neglecting these insignificant terms on the joint density estimate must be quantified or at least bounded.

As an example, we describe the use of the GPB class of algorithms to aid in the suboptimal computation of this joint density. The joint density (11) can be approximated by conditioning only on the most recent input state,

$$\hat{p}(Y(k)) = p(y_1)\prod_{j=2}^{k}\sum_{l=1}^{M}\hat{p}(y_j \mid Y(j-1), u_{j-1}=\gamma_l)\cdot \hat{P}(u_{j-1}=\gamma_l \mid Y(j-1)). \tag{12}$$

In GPB1, the density on the state, $p(x_k \mid Y(k))$ is assumed normally distributed with a mean $\hat{x}_k$ and covariance P_k when in fact it is a sum of M^k separate gaussian densities [4]. Under this assumption, the posterior density on the present observation conditioned on the most recent input state is also gaussian,

$$\hat{p}(y_{k+1} \mid Y(k), u_k=\gamma_m) \sim \mathcal{N}(CA\hat{x}_k + C\gamma_m, C[AP_kA^T+Q]C^T+R). \tag{13}$$

where $\mathcal{N}(m, K)$ denotes a gaussian with a mean vector m and covariance matrix K. The weighting factor $\hat{P}(u_k = \gamma_m \mid Y(k))$ is immediately available from $\hat{P}(u_{k-1}=\gamma_l \mid Y(k))$, $l=1,2,\ldots,M$ and elements of the transition matrix Λ. The values of $\hat{P}(u_{k-1}=\gamma_l \mid Y(k))$ are available from

$$\hat{P}(u_{k-1}=\gamma_l \mid Y(k)) = \hat{p}(y_k \mid u_{k-1}=\gamma_l, Y(k-1))\cdot \frac{\hat{P}(u_{k-1}=\gamma_l \mid Y(k-1))}{\hat{p}(y_k \mid Y(k-1))}, \tag{14}$$

thus completing the recursive weight computation. For more details of the GPB1 algorithm, see [4].

Both terms in the suboptimal likelihood are byproducts of the GPB1 algorithm. Obviously, this algorithm limits the computational complexity of the joint density on the observations by considering only $N=M$ gaussians in the mixture instead of M^k using the hypothesis reduction technique from the GPB1 algorithm. This method can be easily extended to GPBn by considering the most recent n input states which reduces the mixture to $N=M^n$ gaussians.

5. Performance

A quantitative measure of performance is difficult in two respects. First, computation of the true joint density is intractable for large k. Second, performance is a nonlinear function of the observations and thus cannot be determined *a priori*. Devroye and Gyorfi [10] suggest the L_1 norm,

$$\int \mid p-\hat{p}\mid, \tag{15}$$

where $\hat{p}$ is the suboptimal joint density and p is the true density, as a measure of the "distance" between two densities. At each time step, the performance of the GPB algorithm is quantified by using an upper bound on this norm as the "distance" between the true density on y_k and the suboptimal density on y_k. At time step k, the true density on the most recent observation is,

$$p = p(y_k \mid Y(k-1)) = \sum_{j=1}^{M^k}\pi_{k,j}\,g(y_k, m_j) \tag{16}$$

where $\pi_{k,j} = P(U_j(k-1) \mid Y(k-1))$ and $g(y_k, m_j)$ is the gaussian density $p(y_k \mid U_j(k-1))$ parameterized by its mean m_j,

$$g(y_k, m_j) \sim \mathcal{N}(C\sum_{l=1}^{k-1}A^{k-l}u_{l-1}, C[AP_{k-1}A^T+Q]C^T+R). \tag{17}$$

Note that the covariance matrix, which we denote by K_k, is data independent and identical for each gaussian density. The suboptimal density as approximated by the GPB1 algorithm is represented by

$$\hat{p} = \hat{p}(y_k \mid Y(k-1)) = \sum_{i=1}^{M}\hat{\pi}_{k,i}\,g(y_k, \hat{m}_i) \tag{18}$$

where $\hat{\pi}_{k,i} = \hat{P}(u_{k-1}=\gamma_i \mid Y(k-1))$ and $g(y_k, \hat{m}_i)$ is the gaussian density $\hat{p}(y_k \mid u_{k-1}=\gamma_i, Y(k-1))$ parametrized by its mean $\hat{m}_i$ shown in (13). The posterior weights $\pi_{k,j}$ and corresponding means m_j are partitioned such that $\| \hat{m}_i - m_j \|$ is minimized over i; the j^{th} term is then assigned to the i^{th} partition denoted by Ξ_i. Ξ_i corresponds to the i^{th} mean of the suboptimal gaussian mixture. Then the L_1 norm can be written as

$$\int \mid p-\hat{p}\mid = \int\left|\sum_{i=1}^{M}\hat{\pi}_{k,i}\,g(y_k,\hat{m}_i) - \sum_{i=1}^{M}\sum_{j\in\Xi_i}\pi_{k,j}\,g(y_k,m_j)\right|dy_k. \tag{19}$$

This can be upper bounded by

$$\sum_{i=1}^{M}\int\left|\hat{\pi}_{k,i}\,g(y_k,\hat{m}_i) - \sum_{j\in\Xi_i}\pi_{k,j}\,g(y_k,m_j)\right|dy_k. \tag{20}$$

The suboptimal posterior weights are redistributed according to

$$\tilde{\pi}_{k,j} = \frac{\pi_{k,j}}{\sum_{j\in\Xi_i}\pi_{k,j}}\hat{\pi}_{k,i}. \tag{21}$$

Then (20) equals

$$\sum_{i=1}^{M}\int\left|\sum_{j\in\Xi_i}(\tilde{\pi}_{k,j}\,g(y_k,\hat{m}_i) - \pi_{k,j}\,g(y_k,m_j))\right|dy_k \tag{22}$$

$$\leq \sum_{i=1}^{M}\sum_{j\in\Xi_i}\int|\tilde{\pi}_{k,j}\,g(y_k,\hat{m}_i) - \pi_{k,j}\,g(y_k,m_j)|\,dy_k \tag{23}$$

The difference in the gaussians in (23) can be quantified as follows. Let

$$A = \left\{y_k : \frac{g(y_k,m_j)}{g(y_k,\hat{m}_i)} \geq \frac{\tilde{\pi}_{k,j}}{\pi_{k,j}} = \beta_j\right\} \tag{24}$$

The j^{th} term in (23) can be expressed as

$$\int_{R^m-A}(\tilde{\pi}_{k,j}\,g(y_k,\hat{m}_i) - \pi_{k,j}\,g(y_k,m_j))dy_k - \int_A(\tilde{\pi}_{k,j}\,g(y_k,\hat{m}_i) - \pi_{k,j}\,g(y_k,m_j))dy_k, \tag{25}$$

which reduces to

$$2(\pi_{k,j} P_D - \bar{\pi}_{k,j} P_{FA}) + (\bar{\pi}_{k,j} - \pi_{k,j}). \tag{26}$$

The notation P_D and P_{FA} is chosen because these quantities equal the probability of detection and false alarm in a related detection problem. It can be shown that [11]

$$P_D = \mathrm{erfc}\left(\frac{\ln\beta_j - \frac{1}{2}\alpha_{ij}}{\sqrt{\alpha_{ij}}}\right), \tag{27}$$

and,

$$P_{FA} = \mathrm{erfc}\left(\frac{\ln\beta_j + \frac{1}{2}\alpha_{ij}}{\sqrt{\alpha_{ij}}}\right), \tag{28}$$

where

$$\alpha_{ij} = (\hat{m}_i - m_j)^T K_k^{-1} (\hat{m}_i - m_j). \tag{29}$$

Thus the L_1 norm can be upper bounded by

$$2\sum_{i=1}^{M} \sum_{j \in \Xi_i} \left(\pi_{k,j} \mathrm{erfc}\left(\frac{\ln\beta_j - \frac{1}{2}\alpha_{ij}}{\sqrt{\alpha_{ij}}}\right) - \bar{\pi}_{k,j} \mathrm{erfc}\left(\frac{\ln\beta_j + \frac{1}{2}\alpha_{ij}}{\sqrt{\alpha_{ij}}}\right)\right) \tag{30}$$

This bound is a function of the observation data y_k. A data independent bound can be found by taking an expected value over the observation data space. This yields an expected upper bound on the performance of the suboptimal approximator. This bound is small if for each term in (30), $\bar{\pi}_{k,j}$ and $\pi_{k,j}$ are small, or if α_{ij} and $|\bar{\pi}_{k,j} - \pi_{k,j}|$ are both close to zero. Partitions were chosen to keep the α_{ij} as close to zero as possible. This analysis extends to GPBn and a wide variety of other multiple model algorithms that involve hypothesis merging and pruning, including the IMM algorithm.

6. Examples

In this section, we compute the expected value of the upper bound of the L_1 distance (30) by performing Monte Carlo simulations. We average over all possible maneuvers. To use these results in ATR, this expected L_1 bound should be computed for each possible aircraft.

Four scalar linear discrete time examples were chosen to illustrate the effect of system parameters on approximating the density of y_k using the GPB1 algorithm.

$$\begin{aligned} x_{k+1} &= a x_k + u_k + w_k & E[w_k w_j] &= 0.01\delta_{kj} \\ y_k &= c x_k + v_k & E[v_k v_j] &= 0.04\delta_{kj} \\ u_k &\in \Gamma = (\gamma_1, \gamma_2) & E[x_0^2] &= 0.1 \end{aligned} \tag{31}$$

$$\Lambda = \begin{bmatrix} \lambda_1 & \lambda_2 \\ \lambda_2 & \lambda_1 \end{bmatrix}.$$

The initial probability of each input state was 0.5. System parameters take on values as indicated in the table below.

Case	a	c	γ_1	γ_2	λ_1	λ_2
1	0.9	0.5	0.5	-0.5	0.95	0.05
2	0.3	0.5	0.5	-0.5	0.95	0.05
3	0.9	0.5	0.5	-0.5	0.75	0.25
4	0.9	0.5	1.0	-1.0	0.95	0.05

In each case, 60 Monte Carlo simulations were run for each of 1024 possible maneuvers ($k \leq 10$). The averaged L_1 bound was weighted with the maneuver probability (7) to compute the expected bound for each system. Results are shown in Figure 1.

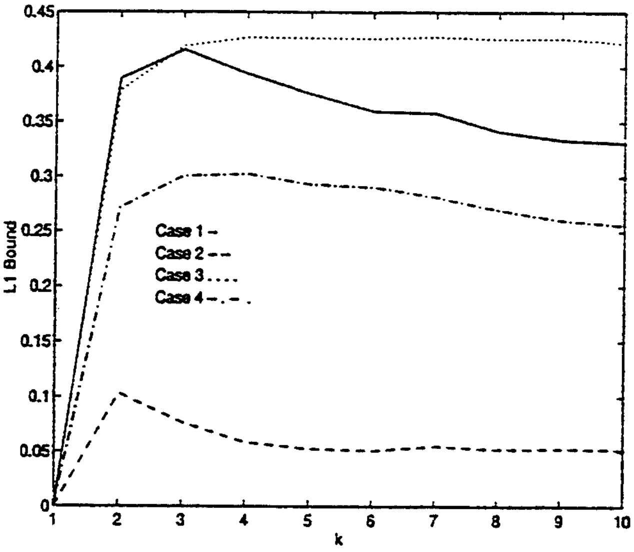


Figure 1: L_1 Bound for GPB1 Algorithm

Case 1 is considered the baseline case. In case 2, the constant a was reduced from 0.9 to 0.3 while keeping the other system parameters constant. The increase in performance is due to the system forgetting the past more quickly than the baseline case. In case 3, the probability of remaining in the same input state was reduced from 0.95 to 0.75. The decrease in performance follows from the increased uncertainty in the system. In case 4, the magnitude of the input state was increased from 0.5 to 1.0. Since the variance of the excitation noises remained constant and the means were spread further, the increase in performance was expected.

It is generally assumed and we believe that increasing the order of the GPB algorithms leads to increased performance. The increase in performance for increasing GPB order depends on system parameters. For fixed order GPB algorithms, increasing the magnitude of system eigenvalues leads to decreasing performance; also increasing the switching rate decreases performance. Obviously, increasing noise variance decreases performance. Quantification of the increase in the expected performance from GPBn to GPB$n+1$ is an ongoing research activity.

7. Conclusions

In this paper, we have developed an ATR methodology that uses prior information on kinematics as a discriminator between target classes. It was shown that computation of the likelihood of target class is an intractable computation. GPB algorithms were suggested for computing suboptimal approximations for conditional output probability density functions. ATR performance is directly related to the accuracy of the suboptimal approximation and a bound on the accuracy of this approximation was derived for a discrete time linear system. Further study includes extending these results to systems modeled by aircraft equations of motion, and quantifying increase in performance for increasing complexity of the suboptimal approximation.

8. Acknowledgment

This research was supported in part by ONR N00014-92-J1418.

9. References

[1] B. Bhanu, "Automatic Target Recognition: State of the Art Survey," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. AES-22, No. 4, pp. 364-379, July 1986.

[2] E. W. Libby, *Application of Sequence Comparison Methods to Multisensor Data Fusion and Target Recognition*. PhD dissertation, Air Force Institute of Technology, 1993.

[3] A. Srivistava, N. Cutaia, J.A. O'Sullivan, M.I. Miller, D.L. Snyder, "Multi-Target Narrow-band Direction Finding and Tracking Based on Motion Dynamics," *Proceedings 30th Annual Conference on Communication, Control, and Computing*, University of Illinois, Urbana-Champaign, 1992.

[4] Guy A. Ackerson, and K. S. Fu, "On State Estimation in Switching Environments." *IEEE Transactions on Automatic Control*, Vol. AC-15, pp. 10-17, February 1970.

[5] J. K. Tugnait and A. H. Haddad, "A Detection-Estimation Scheme for State Estimation in Switching Environments," *Automatica*, Vol. 15, pp. 477-481, 1979.

[6] H. A. P. Blom, and Y. Bar-Shalom, "The Interacting Multiple Model Algorithm for Systems with Markovian Switching Coefficients," *IEEE Transactions on Automatic Control*, Vol. 33, No. 8, pp. 780-783, August 1988.

[7] A. G. Jaffer and S. C. Gupta, "Optimal Sequential Estimation of Discrete Processes with Markov Interrupted Observations," *IEEE Transactions on Automatic Control*, Vol. AC-16, pg. 471, 1970.

[8] Y. Bar-Shalom, editor. *Multitarget-Multisensor Tracking: Advanced Applications, II.*, Norwood, MA: Artech House, 1992.

[9] X. R. Li and Y. Bar-Shalom, "Performance Prediction of the Interacting Multiple Model Algorithm," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 29, pp. 755-771, 1993.

[10] L. Devroye, and L. Gyorfi, *Nonparametric Density Estimation: The L_1 View*. John Wiley & Sons, New York, 1985.

[11] H.L. Van Trees, *Detection, Estimation, and Modulation Theory: Part I*. John Wiley & Sons, New York, 1968.

Appendix C

Table of Contents

K. C. Du, "Range Profile Prediction in Radar Target Recognition," M.S. Thesis, Washington University, September 1993. . . . . .pg C-2

ENGR.803,802

SEVER INSTITUTE OF TECHNOLOGY

Master of Science Degree - Thesis Option

THESIS ACCEPTANCE

(To be the first page of each copy of the thesis)

DATE: August 26, 1993

STUDENT'S NAME: K. Cecilia Du

[redacted]

This student's thesis, entitled Range Profile Prediction in Radar Target Recognition

has been examined by the undersigned committee of three faculty members and has received full approval for acceptance in partial fulfillment of the requirements for the degree MASTER OF SCIENCE.

APPROVAL: Joseph A. O'Sullivan, Chairman

Donald L. Snyder

Distribution:

1 - Department
3 - Thesis Copies
1 - Candidate
1 - Dean's Office
1 - Registrar

WASHINGTON UNIVERSITY
SEVER INSTITUTE OF TECHNOLOGY

Range Profile Prediction in Radar Target Recognition

by

K. Cecilia Du

Prepared under the direction of Dr. J. A. O'Sullivan

A thesis presented to the Sever Institute of
Washington University in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE

December, 1993

Saint Louis, Missouri

WASHINGTON UNIVERSITY

SEVER INSTITUTE OF TECHNOLOGY

ABSTRACT

Range Profile Prediction in Radar Target Recognition

by K. Cecilia Du

ADVISOR : Dr. J. A. O'Sullivan

December, 1993

Saint Louis, Missouri

An algorithm for range profile prediction in radar target recognition is presented. Radar data are assumed to come from two sensors: a tracking radar and a high resolution radar. The tracking radar is used to estimate actual target position and orientation. The high resolution radar collects information in the form of a sequence of measured range profiles. Simulated range profiles are generated based on the estimated target position and orientation from the target tracker and target templates. The target template is a facet model for the surface; the facets are assumed to be electromagnetically noninteracting. These simulated profiles are compared with the measured ones and the target type is estimated. The proposed algorithm for range profile prediction is divided into two parts: generating an illumination mask of the target and computing the reflectivity. Realizations for both tasks are discussed in detail.

TABLE OF CONTENTS

1 Introduction C–9

Rome Lab Radar-Systems facility C–13

Organization C–15

2 Mathematical Background C–16

3 Development of Target Description and Range Profiles C–23

The Coordinate Systems C–23

Illumination Mask C–26

Range Profile C–29

The Model for the Range Profile C–30

An Algorithm Based on the Model C–31

More Processing C–35

4 Issues to Be Addressed C–36

The "sinc" Function C–36

Validation of the Model C–39

5 Results C–43

The Target and the Template C–43

Implementation and the Results C–46

Summary C-53

6 Conclusions and Future Work C-55

7 APPENDIX C-57

8 ACKNOWLEDGEMENTS C-96

9 Bibliography C-97

10 Vita C-99

LIST OF FIGURES

1.1 General Approach for Target Classification. Signals are coming from two sensors: high resolution radar and tracking radar. For the high resolution radar, $s_r(t)$ is the received signal and $w(t)$ is white Gaussian noise. $h(t)$ is shown in chapter 2. . . . . C–12

3.1 Body-centered and inertial-reference coordinate system. C–24

3.2 Simplified shading. If an eye is placed at the -∞ of the X-axis looking into the positive X-direction, then what observer can see is the top of the patch A and patch B. The shadowed part of patch A is shaded because of patch B. C–26

3.3 Quantization of a patch in y direction. C–29

3.4 A patch with 4 normals. C–32

3.5 An Algorithm of Generating Range Profile. C–35

4.1 Error between the true HRR signature and predicted HRR signature C–41

5.1 A faceted model of X-29 with 573 patches displayed on a Silicon Graphics machine, top view. C–44

5.2 A faceted model of X-29 with 573 patches displayed on a Silicon Graphics machine, bottom view. C–45

5.3 A faceted model of X-29 with 573 patches displayed on a Silicon Graphics machine, side view. C–46

5.4 Hashing data organization for the shading mask. The main structure is an array. Each entry of the array is a structured component containing two fields: a real number and a pointer points to a linked list. The linked list has dynamic storage space. C–47

5.5 Range profiles of X-29. Tail Aspect C–49

5.6 Range profiles for the same template with different definition of shaded patch. The crosses are the reflectivity for definition three. The circles are the reflectivity for definition two. The line is the reflectivity for definition one. C–50

5.7 Range profiles for the same template with different ways of modeling the radar cross section. The circles are the reflectivity of the template if the antenna pattern is used. The line is the reflectivity of the template if the squared cosine relation is used. C–51

5.8 Range profiles for the same target with different orientations. There are no shifting and rotations in roll and pitch. The adjacent range profiles differs 1° in yaw. There is only one random phase for all patches. C–52

5.9 Range profiles for the same target with different orientations. There are no shifting and rotations in roll and pitch. The adjacent range profiles differs 1° in yaw. There are different random phases for each patch. C–54

Range Profile Prediction in Radar Target Recognition

1. Introduction

Automatic target recognition (ATR) is the computer processing of data from optical, radar, infrared, or other imaging sensors for classifying a sensed object. An ATR system effectively removes man from the process of target acquisition and recognition. Basically, an ATR system performs automatic target acquisition, identification, and tracking by processing a sequence of data. The algorithmic components of an general ATR system can be decomposed into preprocessing, detection, segmentation, feature computation, selection and classification, prioritization, and tracking [1].

Cohen [2] worked on the problem of ultra-high range resolution radar profiles for target recognition. His main concern is the extreme variability of the target which associates an inordinate amounts of data for training and memory for implementation. Smith and Goggans [3] also worked on the radar target identification problem. In their paper, they give the theory of radar target identification and some simple applications. However, both cases above need a complete set of signatures of all potential targets to be available in the target library. Smith and Goggans mentioned in their work that at least 10^5 signatures per target will be necessary to allow for all possible orientations of a target. The ATR algorithm proposed in this thesis requires only one three dimensional template for every target which saves a tremendous amount of memory space.

In this project, our main concern is noncooperative air targets, which are targets that do not participate (or cooperate) in the process of their identification. Some properties of air targets are integrated into the modeling process. The goal of our ATR system is to decide the number of targets in the scene, to classify the targets, and to track the targets. An algorithm is proposed. The algorithm may be categorized as identification-by-simulation (IBS). IBS means that estimates of the time-varying target position and orientation are used to obtain a real-time simulation of the target return. This simulated return is compared to the actual return to identify targets.

The general approach of the project is shown in Fig. 1.1. It is assumed that the targets are rigid body objects, and can be represented by templates. The templates consist of surface descriptions of the targets of interest. It is also assumed that the surface of a target is partitioned into patches. For example, the templates used in this project contain the corner positions and outward normals of the polygons in the facet representation of the target surface. These patches are electromagnetically large, but physically small. For example, the carrier frequency of the signal used in the project is 3 GHz. The wavelength λ equals

$$\lambda = \frac{c}{f} = \frac{3 \times 10^8 m/s}{3 \times 10^9 Hz} = 10 \quad cm. \tag{1.1}$$

The airplane we used in the simulation is 10 meters long, and has approximately 500 patches. Hence each patch is approximately 1 meter long which extends about 10 wavelengths. This is why we say that a patch is physically small (~ 1 m) and electromagnetically large (covers ~ 10 wavelengths). Electromagnetic interactions between the patches are not accommodated, which is a limitation. However, the templates do account for the distributed nature of the targets.

From Fig. 1.1, we see that data come from two sensors: a high range resolution radar denoted as "HRR" and a tracking radar denoted as "Tracking". The tracking radar collects information over time on the target dynamics. The purpose of using it is to get estimated target orientation, location, and bulk velocity. Then the estimates are applied to the template of the target stored in the 3-D memory. Radar reflections are then synthesized, and the result is a simulation of the target return. Data received from the high resolution radar are sent to a stretch processor. Define the true range profile as the reflectivity of a real target as a function of two-way delay. It is a physical property of the target and it is parameterized by target orientation and position. Define the predicted range profile as the range profile which is generated using our system. In Fig. 1.1, it is the output of the 3-D memory. Define the true HRR signature as the signal part of the output of the stretch processor. Define the predicted HRR signature as the output of the filter whose impulse response is shown as $h(t)$ in Fig. 1.1.

This thesis describes a method of taking the template of a target as input, generating an illumination mask, and computing the range profiles. An illumination mask is a collection of patches that are not shadowed by other patches that are in

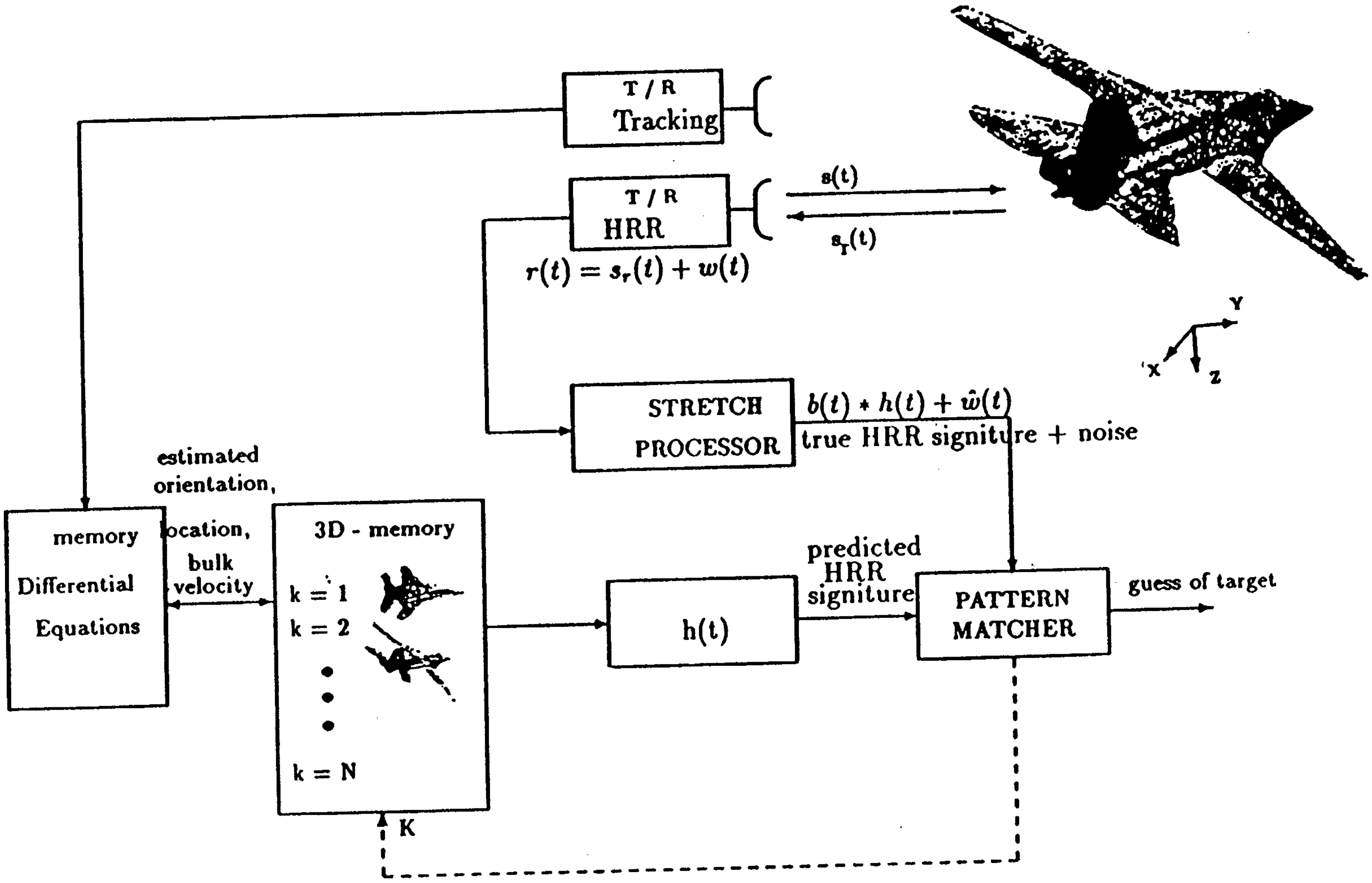


Figure 1.1: General Approach for Target Classification. Signals are coming from two sensors: high resolution radar and tracking radar. For the high resolution radar, $s_r(t)$ is the received signal and $w(t)$ is white Gaussian noise. $h(t)$ is shown in chapter 2.

front of them along the radar line-of-sight. In another words, an illumination mask is a collection of patches that are directly illuminated by the transmitted signal. Many details need to be taken into account for projecting a three dimensional object onto a two dimensional grid and separating the patches that contribute to the returns from the ones that do not. A solution is discussed in Chapter 3. Several difficult issues in simulating the range profiles are addressed. The first issue is how to model the reflectivity of a patch. The second issue is how to deal with quadrilateral patches that are not coplanar. The third issue is how to model the amplitude of the reflected signal from the cross sections of the patches. These problems will also be explained in Chapter 3.

In order to predict the target type, the likelihood function of the received data given the target type is computed. The discrimination between the targets is made in the sense of maximizing the likelihood function. In our case, since the noise is white, maximizing the likelihood is equivalent to minimizing the mean square error between the predicted HRR signature and the true HRR signature with additive white Gaussian noise.

1.1 Rome Lab Radar-Systems facility

The radar frequency-spectrum is divided into various bands. Each band has its own designation and usage. Rome Laboratory has three ground-based radar systems: L-Band, S-Band, and C-Band radar.

The L-Band radar:

use:	tracking radar
carrier frequency:	$f_c = 1{,}265$ MHz

wavelength: $\lambda = 23.7$ cm

bandwidth: $B = 20$ MHZ

beamwidth: 2.5° in azimuth 8° in elevation

scan rate: 0 - 10 rpm

pulse duration: up to 600 μs

pulse repetition-rate: 360 pps

range resolution: 7.5 m

The S-Band radar:

use: tracking radar
radar cross-section identification
radar target recognition

carrier frequency: $f_c = 3{,}350$ MHz

wavelength: $\lambda = 9$ cm

bandwidth: $B = 2.5$ MHz for tracking
$B = 320$ MHz for target recognition

beamwidth: 1.2° in azimuth and elevation

scan rate: no info is available

pulse duration: up to 600 μs

pulse repetition-rate: up to 6% duty cycle

range resolution [1]: $\Delta R = \frac{3 \times 10^8}{2 \times \tau\beta}$

The C-Band radar:

---

[1] In the normal range-resolution mode, $\tau = 40\ \mu$ s, and $\tau\beta = 2.5$ MHz, yielding $\Delta R = 60$ m, where τ is the chirp pulse duration, and β is the chirp pulse slope. In the ultra-high bandwidth mode, $\tau\beta = 320$ MHz, yielding $\Delta R = 0.88$ m.

use: tracking radar

carrier frequency: f_c = 5,650 - 5,900 MHz

wavelength: λ = 5.1 - 5.3 cm

bandwidth: B = 10 MHZ

beamwidth: 1° in azimuth 2° in elevation

In this project, we assume that the S-Band radar is used in the radar target recognition model.

1.2 Organization

The organization of the thesis is as follows. Chapter 2 gives some mathematical background. Chapter 3 describes the development of range profiles using the shape model. Chapter 4 addresses some issues in the algorithm. Chapter 5 presents the results. Chapter 6 summarizes the work and programs are listed in the appendix.

The contribution of this thesis is an algorithm to generate the range profile of a noncooperative target. It plays an important role in the whole project because the goal of the project is to estimate the target type of an object. The maximum likelihood method is used to determine the estimate. In order to compute the likelihood of the received data given target type, the range profile is a key part. It is important also because the returned signal is not available to us right now. In order to simulate the returned signal, the range profile is necessary.

2. Mathematical Background

"Matched filter" is a frequently used term in radar signal analysis. A filter matched to a given signal is an optimal filter for reception of that signal in additive white Gaussian noise. The filter is optimum in several senses including maximizing the output signal to noise ratio and finding the maximum likelihood of the received data given some parameters [4][5].

The motivation to use the matched filter in this project is mainly the likelihood ratio criterion. The received signal, $r(t)$, has the form

$$r(t) = s(t, A) + w(t), \quad 0 \leq t \leq T,$$

where $s(t, A)$ is the signal part, and is parameterized by A; $w(t)$ is white Gaussian noise with intensity N_0. A is the parameter to be estimated. The null received signal is white Gaussian noise with intensity N_0. The observation $r(t)$ is a time-continuous random waveform. The log likelihood ratio of the received signal [5] is

$$l[r(t), A] = \frac{2}{N_0}\int_0^T r(t)s(t, A)dt - \frac{1}{N_0}\int_0^T s^2(t, A)dt. \tag{2.1}$$

If the signals are complex, then

$$l[r(t), A] = \frac{1}{N_0}Re\{\int_0^T [2r(t) - s(t, A)]s^*(t, A)dt\}. \tag{2.2}$$

For an estimation problem, the optimal estimate of A is the value that maximizes (2.1) or (2.2). Note that the first part of (2.1) or (2.2) is the received signal $r(t)$ correlated with a copy of its signal part. The matched filter is an implementation of this for each choice of A.

The characteristic of matched filter can be designated by either a time response function

$$h(t) = s^*(-t), \tag{2.3}$$

or a frequency response function

$$H(w) = S^*(w), \tag{2.4}$$

where $s(t)$ is the signal part of the input signal to the filter, $S^*(w)$ is the Fourier transform of $s^*(-t)$.

In this project, the high resolution radar(HRR) is assumed to transmit a sequence of chirp pulses, one of which is

$$s_H(t) = exp[j2\pi(f_0t + kt^2/2)], \quad 0 \leq t \leq T, \tag{2.5}$$

where T is the signal duration, in seconds, and kT is the chirp bandwidth, in hertz. The subscript H denotes HRR. The received signal equals

$$r(t) = \int_{-\infty}^{\infty} s_H(t-\tau)b(t-\tau/2,\tau)d\tau + w(t), \tag{2.6}$$

where w is complex white Gaussian noise with intensity N_0, and $b(t,\tau)$ is the total reflectivity density of the target at two-way delay τ at time t. That is $b(t,\tau)$ equals the integral of the reflectivity density on the surface of the target of points at two-way delay τ, at time t. It is understood that only the patches that are directly illuminated by the transmitted signal contribute to the range profile. This is accommodated in $b(t,\tau)$.

Assume that the target does not rotate significantly during an illumination, then the range profile for a pulse at time T' equals $b(t-\tau/2,\tau) \approx b_{T'}(\tau)$, for t in the neighborhood of T'. The generation of the range profiles is discussed in the next chapter. Substituting this into (2.6), the received signal from HRR can be modeled as

$$r(t) = \int_{-\infty}^{\infty} s_H(t-\tau) b_{T'}(\tau) d\tau + w(t). \tag{2.7}$$

Typically a sequence of such waveforms is obtained. Suppose the transmitted signal is

$$\sum_m s_H(t-mT'). \tag{2.8}$$

The m^{th} received signal from HRR is

$$r_m(t) = \int_{-\infty}^{\infty} s_H(t-\tau) b_{mT'}(\tau) d\tau + w_m(t) \tag{2.9}$$

$$= s_H(t) * b_{mT'}(t) + w_m(t), \tag{2.10}$$

where $b_{mT'}(\tau)$ is the corresponding range profile, and $*$ denotes convolution.

The filter used in the project almost performs the matched filter-processing. It is not exact because the filter convolves the received signal with the output of a local oscillator,

$$exp[j2\pi(f_0 t - kt^2/2)], \tag{2.11}$$

which is the complex conjugate of the transmitted chirp, over a time period, $\tilde{T}$, much larger than the pulse duration T. $\tilde{T}$ is the time duration of the chirp used for the filter. This type of "matched" filter is also called a "stretch processor". If $\tilde{T}$ were equal to T, then the filter would be the matched filter, and the output of the filter would be a sufficient statistic of the likelihood function of the received data. Ideally, one would implement a matched filter instead of the stretch processor. The stretch processor is used because at Rome Laboratory it is a part of the existing facilities. This algorithm is designed for Rome Laboratory and eventually will be used there; we are not assuming hardware modification is possible.

The output of the stretch processor is determined by two terms: a signal term, namely the true HRR signature, and a noise term. The signal part equals the convolution of the first term from (2.10) with the chirp (2.11). Because of the associative and commutative properties of the convolution,

$$
\begin{aligned}
&(s_H(t) * b_{mT'}(t)) * exp[j2\pi(f_0 t - kt^2/2)] \\
&= b_{mT'}(t) * (s_H(t) * exp[j2\pi(f_0 t - kt^2/2)]).
\end{aligned}
$$

Substituting (2.5) for s_H and defining $sinc(x) = sin(\pi x)/(\pi x)$, we have

$$
\begin{aligned}
h(t) &= s_H(t) * exp[j2\pi(f_0 t - kt^2/2)] && (2.12) \\
&= \int_0^T exp[j2\pi(f_0\tau + k\tau^2/2)]exp[-j2\pi(f_0(\tau - t) + k(\tau - t)^2/2)]d\tau && (2.13) \\
&= exp[j2\pi(f_0 - kt^2/2)] \int_0^T exp[j2\pi kt\tau]d\tau && (2.14) \\
&= \frac{1}{j2\pi kt} exp[j2\pi(f_0 t - kt^2/2)][exp(j2\pi kTt) - 1] && (2.15) \\
&= exp[j2\pi(f_0 t - k(t^2 - Tt)/2)]sinc(kTt). && (2.16)
\end{aligned}
$$

The noise part equals the convolution of the noise term $w_m(t)$ from received signal with the chirp in (2.11). If the $w_m(t)$ is assumed to be white, then the resulting noise $\hat{w}_m(t)$ is approximately white. It is approximately white because the Fourier transform of a chirp is a chirp which has magnitude 1, and the chirp we used has a relatively large bandwidth. If the bandwidth of the chirp is infinite, then $\hat{w}_m(t)$ is white.

Combining the two parts, the output of the stretch processor from the m^{th} transmitted signal is

$$d_{mT'}(t) = \int_{-\infty}^{\infty} h(t-\tau) b_{mT'}(\tau) d\tau + \hat{w}(t). \tag{2.17}$$

This motivates us to compute the log likelihood function of the stretch processor output given the range profile. Given the range profile, the computation required is

$$l(d|b) = \frac{1}{N_0} Re\{\sum_m \int_{-\infty}^{\infty} [2d_{mT'}(t) - h(t-\tau) b_{mT'}(\tau) d\tau][\int_{-\infty}^{\infty} h(t-\tau) b_{mT'}(\tau) d\tau]^* dt\}. \tag{2.18}$$

Then the next question is what to do with this log likelihood function. The final goal is to compute the likelihood of a given set of data for different target types. It is determined by the prior on the orientations and positions, the tacking data, and the model for the reflectivity density, $b_{T'}(t)$. Let y_s denote the received tracking data with the argument t suppressed, b denote $b_{T'}(t)$ with the arguments suppressed, and i denote the target type, assuming the data y_s and b are conditionally independent given the position and orientation of the target, x, a joint likelihood is

$$L(d, b, y_s|\mathrm{x}, i) = L(d|b) L(b|\mathrm{x}, i) L(y_s|\mathrm{x}, i). \tag{2.19}$$

Incorporating the likelihood on x and a prior on i,

$$L(d, b, y_s, \mathbf{x}, i) = L(d|b)L(b|\mathbf{x}, i)L(y_s|\mathbf{x}, i)L(\mathbf{x}|i)P_i. \tag{2.20}$$

The computations required for the first factor, $L(d|b)$, are shown in (2.18). For the second factor, $L(b|\mathbf{x}, i)$, the model for $b(\tau)$ given target type and target position and orientation will be described in next chapter. Once a model is derived, computing the likelihood function is straightforward. The likelihood on x given target type i and tracking data y_s given target type i and its orientation x and are determined by the dynamics of the target type and are governed by nonlinear differential equations. P_i is the prior on the target type, and $\sum P_i = 1$. N. Cutaia [6] has been working on modeling the target dynamics, and A. Srivastava [7] has been working on target tracking. Note that in (2.20), b and y_s are observed data; i is the parameter to be estimated. The functions b and x are parameters that, if known, would improve the performance of the identification algorithm. The idea is that if the likelihoods are fairly peaked, the actual functions are in the neighborhood of the MAP estimates for them. Thus, the MAP estimates are used for these functions. A proposed target identification algorithm has the form

$$\hat{i}_{MAP} = \arg\max L(d, y_s, \hat{b}_{MAP}(y_s, d|\hat{\mathbf{x}}_{MAP}, i), \hat{\mathbf{x}}_{MAP}(y_s, d|i), i). \tag{2.21}$$

3. Development of Target Description and Range Profiles

In this chapter, we emphasize the generation of the illumination mask and the range profiles of a target. First, different coordinate systems and their relationship are introduced. Second, an algorithm for generating the illumination mask of a target is proposed. Third, a model for the reflectivity of a target is presented. Last, an implementation of the model is discussed.

3.1 The Coordinate Systems

To describe the orientation of an object in space, the coordinate system that is used needs to be specified. In our project, there are two types of coordinates involved: a body-centered frame and an inertial-reference frame.

As the name suggests, a body-centered frame is a coordinate system whose origin is placed at the center of mass of the rigid body. As the main concern of this project is air targets, the conventional aircraft definition of a body-centered coordinate system is used. Define X_B, Y_B, and Z_B to be the three axes, where X_B points from the origin to the nose of the aircraft; Y_B points from the origin to the right wing of the aircraft; and Z_B points down. They are shown geometrically in Fig. 3.1.

An inertial-reference frame contains axes which are not accelerating or rotating. In the project, the origin of this coordinate system is located at the radar. Let

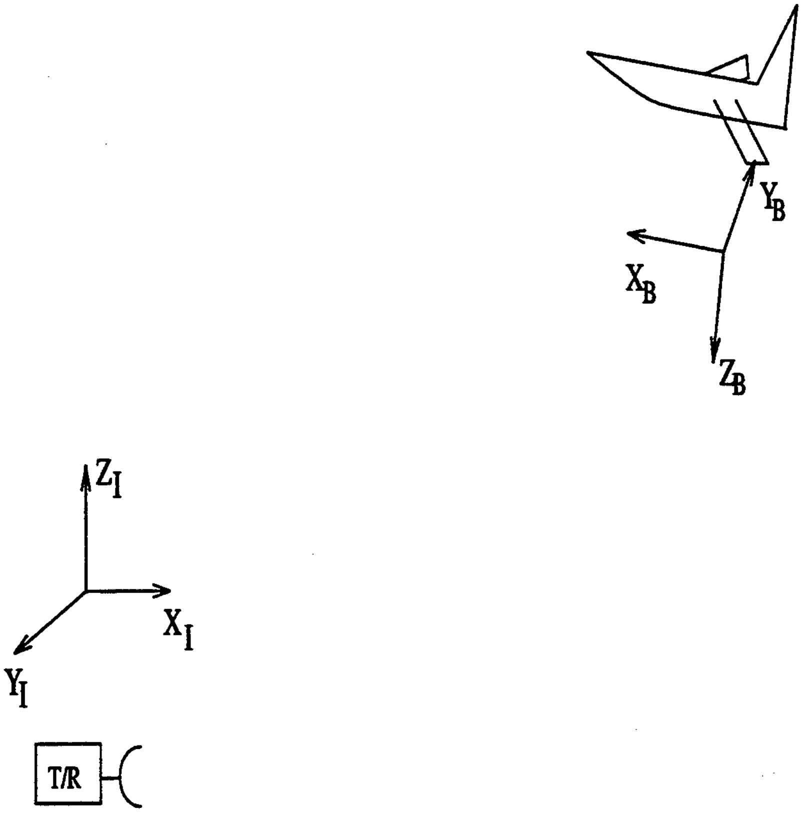


Figure 3.1: Body-centered and inertial-reference coordinate system.

X_I, Y_I, and Z_I be the axes defined as the usual right-hand coordinates in three dimensions, with the X_I axis pointing in the direction of the transmitted signal.

The two coordinate systems are related by a rotation matrix and a translation vector. Suppose we know that there is a target positioned at (a_I, b_I, c_I) in the inertial coordinates and a unit vector along the line-of-sight from the radar to the target has angles (ψ, θ, ϕ) with respect to the X_I, Y_I, Z_I axes. In our three dimensional memory, the template of the target is given in the body-centered frame. How do we simulate the target using the given template? The idea is to rotate the template and then translate it to the proper position. First we will define some terminology. Let X denote North; Y denote East; Z denote the direction towards earth center. Let x denote forward; y denote right; and z denote down in the body centered frame looking towards the plane nose. Then, roll is an angle in the yz plane (right and down). It is measured from the intersection of horizontal and yz plane to y, positive clockwise looking in x direction. Pitch is the angle measured from the XY plane to the x axis, positive when x is elevated above the horizontal plane. Sometimes it is also called head-up attitude. Yaw is an angle in the XY plane. It is measured from X to the projection of x, positive clockwise looking in Z direction. Then the sequence of the transformations should be

First, rotate the templates ψ (yaw) about the z axis

Second, rotate the templates θ (pitch) about the y axis

Third, rotate the templates ϕ (roll) about the x axis

Fourth, shift the templates to (a_I, b_I, c_I)

Mathematically, combining the first three steps, we have the total rotation matrix $\mathbf{R}(\phi, \theta, \psi)$ [8]

$$\mathbf{R}(\phi,\theta,\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & sin\phi \\ 0 & -sin\phi & cos\phi \end{bmatrix} \begin{bmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{bmatrix} \begin{bmatrix} cos\psi & sin\psi & 0 \\ -sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.1)$$

$$= \begin{bmatrix} cos\theta cos\psi & sin\phi sin\theta cos\psi - cos\phi sin\psi & cos\phi sin\theta cos\psi + sin\phi sin\psi \\ cos\theta sin\psi & sin\phi sin\theta sin\psi + cos\phi cos\psi & cos\phi sin\theta sin\psi - sin\phi cos\psi \\ -sin\theta & sin\phi cos\theta & cos\phi cos\theta \end{bmatrix}^T . \quad (3.2)$$

The total transformation is

$$\mathbf{x}_i = \mathbf{R}(\phi,\theta,\psi)\mathbf{x}_b + (a,b,c)^T. \quad (3.3)$$

3.2 Illumination Mask

If the surface of a target can be partitioned into patches, the illumination mask is a collection of patches that are directly illuminated by the transmitted signal. To generate a illumination mask, the patches that are shaded by the ones in front of them need to be separated from the ones that are not. How do we discriminate between the patches?

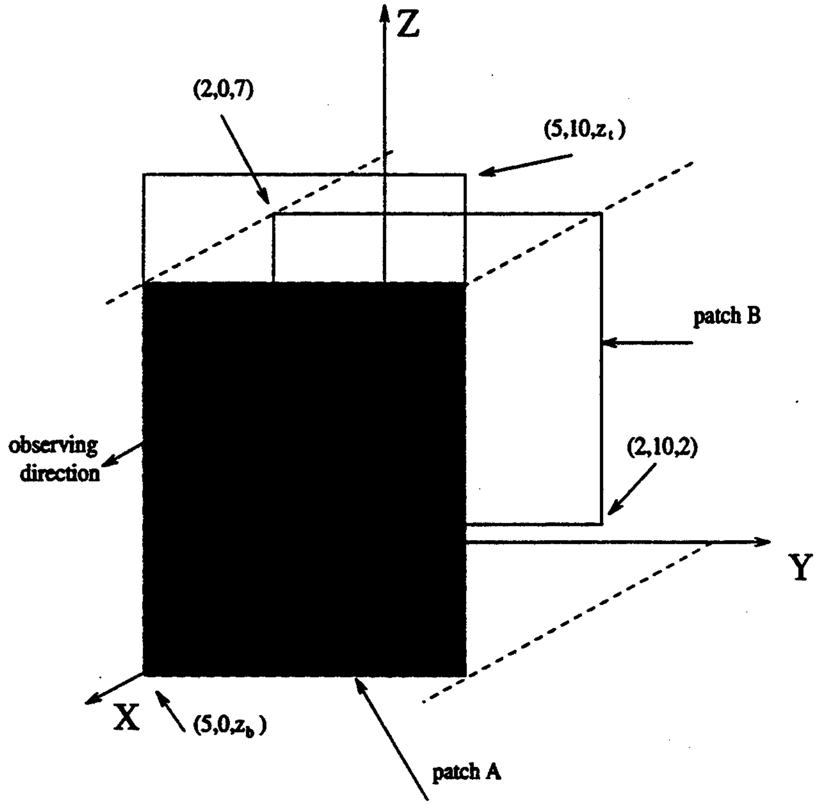


Figure 3.2: Simplified shading. If an eye is placed at the -∞ of the X-axis looking into the positive X-direction, then what observer can see is the top of the patch A and patch B. The shadowed part of patch A is shaded because of patch B.

First, a simplified problem is presented. Assume there are two plates, A and B, placed in the typical Cartesian coordinate system as shown in Fig. 3.2. Plate A and B are placed in such a way that they are parallel to each other and the YZ plane. The vertices of plate A are (5, 0, z_b), (5, 10, z_b), (5, 10, z_t), and (5, 0, z_t). The vertices of plate B are (2, 0, 2), (2, 10, 2), (2, 10, 7), and (2, 0, 7). The observing point is at the origin and looking in the X direction. Also, an orthographic projection is assumed. The illumination mask M of this simple case depends on the values of z_b and z_t. There are six possibilities:

$$\begin{aligned} case1 &: \; z_b < z_t \leq 2 \\ case2 &: \; z_b < 2 \text{ and } 2 < z_t < 7 \\ case3 &: \; 2 \leq z_b \leq 7 \text{ and } 2 \leq z_t \leq 7 \\ case4 &: \; 2 \leq z_b \leq 7 \text{ and } z_t > 7 \\ case5 &: \; 7 < z_b < z_t \\ case6 &: \; 2 < z_b \text{ and } 7 < z_t \end{aligned}$$

The conclusion we should get out from this example is that a illumination mask is parameterized by the x values, and the minimum and the maximum z value at a specific x. And when the problem gets complicated, the mask probably will depend on more parameters. In the implementation, a linked list is used. Each node of the list contains the three parameters, x, zmin, and zmax. For case 2 of this example, there will be two nodes in the list. The x value for the first node is 2; the minimum z value is 2; and the maximum z value is 7. The x value for the second node is 5; the minimum z value is 0; and the maximum z value is 2. Obviously, this linked list captures all the information needed to define the illumination mask for this two-patch case. For case 3, there will be only one node as plate A is fully shaded by plate B. And for case 6, there will be three nodes in the list. The x value for the first node is 2; the minimum z value is 2; and the maximum z value is 7. The x value for the second node is 5; the minimum z value is z_b; and the maximum z value is 2. The x value for the second node is also 5; the minimum z value is 7; and the maximum value is z_t.

The next step towards solving the illumination mask problem is to generalize the simple case above. There are two major differences between the general problem and the case above. The first one is that the x values for the template are continuous instead of discrete. The second one is that the template is much more complicated than the patches used above in the Y direction. Temporarily assume every patch is flat. To decompose this complicated problem into the simple problem above, we find the maximum and minimum y value; divide ymax - ymin into one hundred equal sized bins. Then each patch is assigned to the appropriate bins according to its y values. Fig. 3.3 shows an example. To visualize this process, one can image that a columnwise partitioned grid is pushed through the x direction of the template. The x value of the patch is determined by the corners of the patch. Let's assume that for each bin, x values for that part of the patch are the same. This assumption is released later. For each patch, the minimum and maximum z values associated with each bin are calculated for the corresponding y value. Now, for each bin, we have a problem that is similar to the simple case above. The equations used to calculate the z values and the x values are

$$z = corneri.z + \frac{(y_i - corneri.y)(corner[i+1].z - corneri.z)}{corner[i+1].y - corneri.y}, \tag{3.4}$$

$$x = corneri.x + \frac{(y_i - corneri.y)(corner[i+1].x - corneri.x)}{corner[i+1].y - corneri.y}, \tag{3.5}$$

where, y_i is the middle value of bin i in the y direction; corneri and corner[i+1] are the adjacent corners of y_i in the y direction.

Combining all the bins, or the linked lists in the implementation sense, we get the illumination mask for the template.

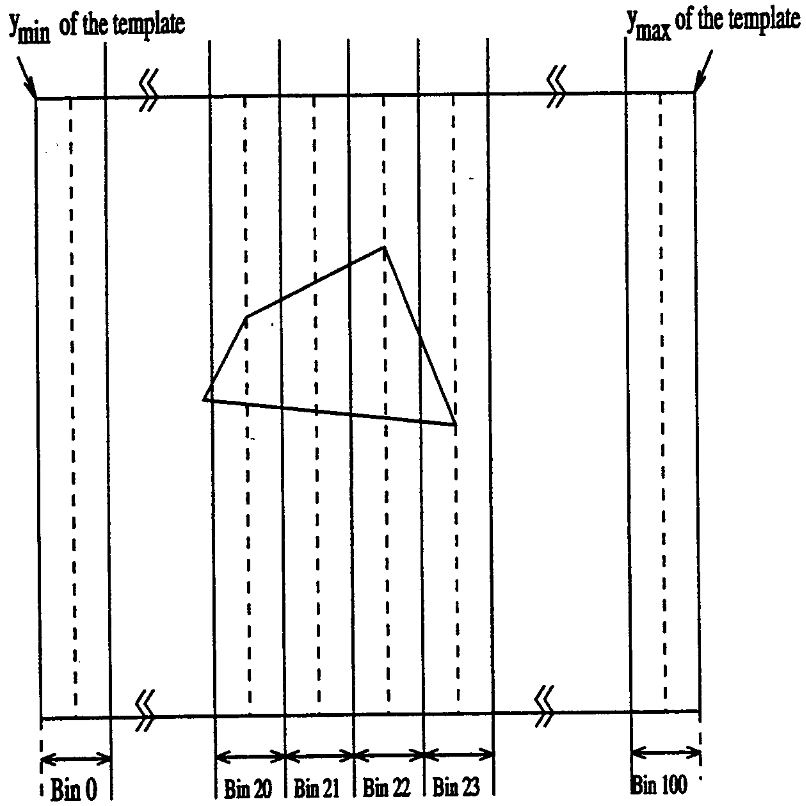


Figure 3.3: Quantization of a patch in y direction.

3.3 Range Profile

In this section, a model of the reflectivity for a target is presented. An algorithm of generating the range profiles based on the model is derived. This model is based on physical optics approximations as presented in the literature [9]. It is a model for the amplitude at the received signal so it incorporates the phase. However, the limitation of this model is that the method does not yield precise signatures but that any more accurate method can be embedded in the ATR algorithm in its place. The method has an advantage of being relatively fast and possibly adequate for some ATR situation. In the literature, Andersh, et al., [10] have been working on a similar problem of synthesizing range profiles for noncooperative target identification. They generated a large number of high resolution range profiles of realistic targets using XPATCH. XPATCH is a software package that is implemented to calculate range profiles from the facet target based on the shooting and bouncing ray technique. Andersh, et al. were able to compare the synthesized range profiles with measured data to demonstrate agreement.

3.3.1 The Model for the Range Profile

We know the reflectivity of a point target can be modeled as a combination of a real part and an imaginary part. The magnitude corresponds to the physical properties of the particular airplane such as surface material and the angle of the incident signal with respect to the normal vectors. The phase is due in part to the fact that the bulk distance is not known to within a wavelength. For a spatially extended target, we integrate the reflectivity of a point target over the extended region. Our model approximates the integral by a sum. So the reflectivity has the form

$$b(\tau) = \sum_k \sum_i \beta g_{k,i} f(\xi_{k,i}, A_{k,i}) e^{j2\pi\theta_k} \delta(\tau - \frac{2x_{k,i}}{c}). \tag{3.6}$$

Here

τ denotes the range position of the target;

b is a constant;

$g_{k,i}$ denotes the illumination mask of i^{th} corner of the k^{th} patch;

$f(\cdot)$ denotes an appropriate function;

θ_k is a random angle.

$A_{k,i}$ is the area associated with i^{th} corner of the k^{th} patch.

One of the difficulties with the model is the choice of the distribution for θ_k. At one extreme it may be uniformly distributed from $-\pi$ to π. At the other extreme it could be determined completely by the position in the X_I direction. In between it is the sum of a constant angle determined by its position and a random component.

3.3.2 An Algorithm Based on the Model

We have talked about different coordinate systems, and how to generate an illumination mask. We also mentioned that the available parameters are the templates stored in the 3-D memory and the estimates of orientation and position of a target. How do we connect these building blocks to get the range profile?

In the 3-D memory, a template is characterized by position vectors and direction normal vectors. The template frame using (3.3) and the estimated orientation and position. Denote by $template_I$ the template in the inertial frame. As the patches in the template are quadrilateral and the four corners that define a patch may not be coplanar, each patch is decomposed into four smaller parallelograms associated with each corner as shown in Fig. 3.4. Each parallelogram is assumed to be flat. The normal vector associated with each corner is assumed to be perpendicular to the parallelogram constructed for that corner. Hence, the technique introduced above can be applied to the parallelograms for generating the illumination mask of the target.

From the illumination mask, each patch is assigned to either 1 or 0 to denote whether it is shaded or not. 1 means that a patch is not shaded, and 0 shaded. 1 or

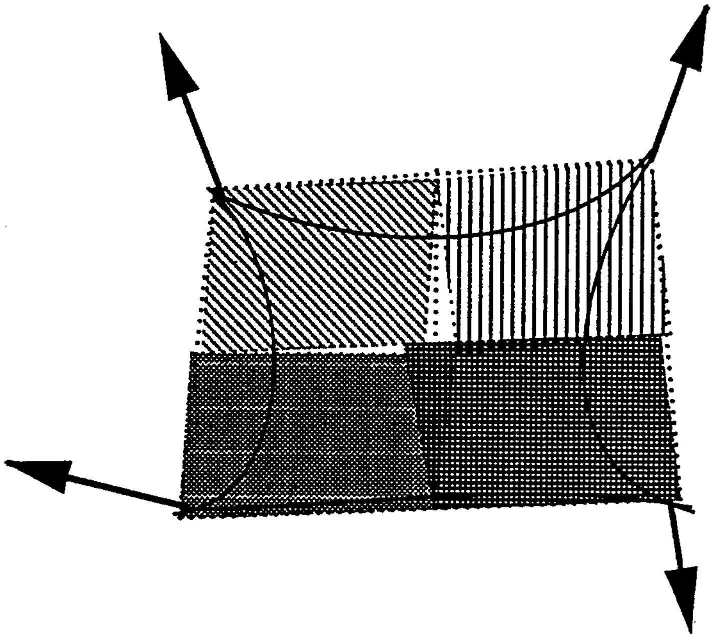


Figure 3.4: A patch with 4 normals.

0 corresponds to the $g_{k,i}$ in (3.6). b is assumed to be 1 in (3.6); it will be assigned to a more accurate value once the physical details of the target are investigated.

The $f(\cdot)$ in (3.6) corresponds to how we model the cross section of the target. It is defined by two parameters: the incident angle and the area. The incident angle is the angle between the incident radar wave and the normal vectors. Let $\mathrm{e} = [-1\ 0\ 0]^T$, then the incident angle is equal to

$$\xi_{k,i} = cos^{-1}(\mathrm{e} \cdot \mathrm{n}_{k,i}), \tag{3.7}$$

where $\mathbf{n}_{k,i}$ is the direction normal vector for the i^{th} corner of the k^{th}patch of $template_I$.

To compute the areas of the regions, areas of the parallelograms are computed. A parallelogram associated with a corner of a patch is constructed in such a way that it is bounded by the two vectors from the corner to the adjacent corners. The area associated with the corner is 1/4 of the cross product of the two vectors. For example, for patch k, the four corners are labeled counter clockwise starting at the lower left corner. For each corner, there are two vectors

$$\mathbf{v}_1 = \mathbf{x}_{I,k,2} - \mathbf{x}_{I,k,1} \qquad \textit{and} \qquad \mathbf{v}_2 = \mathbf{x}_{I,k,2} - \mathbf{x}_{I,k,3} \tag{3.8}$$

associated with it. The parallelogram associated with corner 2 is defined by $\frac{1}{2}\mathbf{v}_1$ and $\frac{1}{2}\mathbf{v}_2$. Hence the area of the region associated with corner 2 of the k^{th} patch is

$$A_{k,2} = \frac{1}{4}\|\mathbf{v_1} \times \mathbf{v_2}\|, \tag{3.9}$$

where

$\times$ denotes vector cross-product;

$\|\cdot\|$ denotes Euclidean norm.

These are several choices for $f(\cdot)$. All the choices have one thing in common: they specify relations between the incident power to an object and its reflecting power at a given observation angle. One choice of $f(\cdot)$ is

$$f(\xi_{k,i}, A_{k,i}) = A_{k,i} \cos \xi_{k,i}. \tag{3.10}$$

(3.10) models a cosine-squared energy dependence on the incidence angle. Another choice of $f(\cdot)$ is an antenna pattern. It has the form of

$$f(\xi_{k,i}, A_{k,i}) = A_{k,i} \frac{J_1(2\pi \sin(\xi_{k,i}) r_{k,i}/\lambda)}{2\pi \sin(\xi_{k,i}) r_{k,i}/\lambda}, \tag{3.11}$$

where $J_1(\cdot)$ is a Bessel function of the first kind, $r_{k,i}$ is found such that $\pi r_{k,i}^2 = A_{k,i}$, and λ is the wavelength of the carrier. This choice corresponds to the antenna pattern of a circular aperture centered at the corner, with area equal to the area of the parallelogram. One implementation detail should be mentioned. At $sin(\xi_{k,i}) = 0$, (3.11) is invalid as $f(\cdot) = \frac{0}{0}$. L'Hosptial's rule is applied and the result is 1/2. $f(\cdot)$ can also be modeled as the backscattering radar cross section using a physical optics approximation in the $\xi_{k,i}$ direction,

$$f(\xi_{k,i}, A_{k,i}) = \frac{A_{k,i}}{\tan \xi_{k,i}} \left[J_1 \left(\frac{4\pi r_{k,i} \sin \xi_{k,i}}{\lambda} \right) \right], \tag{3.12}$$

where λ, $r_{k,i}$, and J_1 are defined as the same in (3.11). The main difference between (3.11) and (3.12) is that (3.12) includes the two-way phase shift as shown by the 4π in the Bessel function instead of 2π in (3.11). There are many other geometric shapes that can be used here. For example, if a thin plate with width W and length L is used instead of the circular disk, then $f(\cdot)$ is

$$f(\xi_{k,i,}, A_{k,i}) = \frac{2\sqrt{\pi} WL}{\lambda} \cos \xi_{k,i} \left[\frac{sin(kW \sin \xi_{k,i})}{kW sin \xi_{k,i}} \right]. \tag{3.13}$$

Other possibilities of $f(\cdot)$ using backscattering radar cross section are discussed in [9]. This is one significant way the processing done to date could be improved. The function $f(\cdot)$ could depend on the patch in the sense that the cross section of the parallelograms could be used to predict the return. This would increase computations correspondingly.

Above, we have discussed the meaning and generation of every component in (3.6). Combining all terms together, we get a synthesized range profile. Fig. 3.5 summarizes the process.

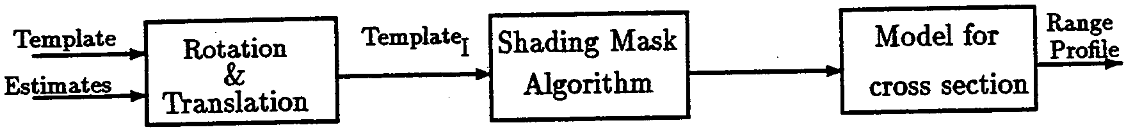


Figure 3.5: An Algorithm of Generating Range Profile.

3.4 More Processing

From Fig. 1.1 and (2.16), we see that the predicted range profile is convolved with the product of a sinc function and a chirp function. However, at the present time, a modified function is used instead of the product. The reason for using the simplified version is explained in the next chapter.

4. Issues to Be Addressed

4.1 The "sinc" Function

In Fig. 1.1, the output of the stretch processor, $d_{T'}(t)$ is expressed in terms of the range profile, a sinc function, a chirp, and the noise part. In order to implement the pattern matcher, we have to get a closed form for $d_{T'}(t)$. We tried two approaches: analytically and numerically.

Analytically, we attacked the problem using some fundamental properties from system theory. In Chapter 2 we derived expressions for $h(t)$ and $d_{T'}(t)$. They have the forms

$$h(t) = exp[j2\pi(f_0 t - k(t^2 - Tt)/2)]sinc(kTt), \tag{4.1}$$

and

$$d_{T'}(t) = \int h(t-\tau) b_{T'}(\tau) d\tau + \tilde{w}(t). \tag{4.2}$$

To simplify the analysis, ignore the noise term in (4.2). The signal part in (4.2) is the convolution of $h(t)$ and a range profile. It is understood that a range profile is a reflectivity profile of a target, and it is a property of the target - reflectivity as a function of 2-way delay. The range profile is not a function of transmitted signal. From the system theory, it is well known that "Convolution in the time domain is equivalent to multiplication in the frequency domain." Therefore, the idea is to take Fourier transforms of $h(t)$ and $b_{T'}(t)$. Let $H(f)$ denote the Fourier transform of $h(t)$, and $B(f)$ denote the Fourier transform of $b_{T'}(t)$. $d_{T'}(t)$ is equivalent to

$$d_{T'}(t) = \mathcal{F}^{-1}\{B(f) \cdot H(f)\}. \tag{4.3}$$

Using the dual property stated above, "multiplication in the time domain is equivalent to the convolution in the frequency domain," we can express H(f) as

$$H(f) = \mathcal{F}\{exp\{j2\pi(f_0 t - k(t^2 - Tt)/2)]\}\} * \mathcal{F}\{sinc(kTt)\}, \tag{4.4}$$

where * denotes convolution. Let

$$q(t) = exp[j2\pi(f_0 + kT/2)t - j2\pi kt^2/2] \tag{4.5}$$

$$= exp[j2\pi(\overline{f}t - kt^2/2], \tag{4.6}$$

where $\overline{f} = f_0 + kT/2$. Hence, $Q(f)$ has the form

$$Q(f) = \mathcal{F}\{q(t)\} \tag{4.7}$$

$$= ae^{j\pi p(f-\overline{f})^2}, \tag{4.8}$$

where a is some constant, and $p = k^{-1}$. These equations can be also found in [4] eq. (4.5) and eq. (4.9). We know that the Fourier transform of a sinc function is a window and has the form

$$G(f) = \mathcal{F}\{sinc(kTt)\} = \begin{cases} \frac{1}{kT} & -\frac{kT}{2} \leq f \leq \frac{kT}{2}, \\ 0 & otherwise. \end{cases}$$

Hence,

$$H(f) = \int_{-\infty}^{\infty} Q(f-\xi)G(\xi)d\xi \tag{4.9}$$

$$= \int_{-\frac{kT}{2}}^{\frac{kT}{2}} bexp[j2\pi p(f-\xi-\overline{f})^2]d\xi, \tag{4.10}$$

where b is some constant which does not depend on ξ. Note that the integral in (4.10) is not easy to solve. We tried various integration techniques, yet the result was not satisfying because of the accuracy.

We then tried to solve the integral using different software packages. The best among these tries is the result we got by using Mathematica package where (4.10) is solved in term of error function[1]. The solution for (4.10) is

$$H(f) = b\frac{(-1)^{\frac{1}{4}}Erf[(-1)^{\frac{3}{4}}(\overline{f}-f-\frac{kT}{2}\sqrt{2\pi p})]}{2\sqrt{2p}}$$
$$-b\frac{(-1)^{\frac{1}{4}}Erf[(-1)^{\frac{3}{4}}(\overline{f}-f+\frac{kT}{2}\sqrt{2\pi p})]}{2\sqrt{2p}}.$$

We also attempted to solve it numerically. To solve (4.2), we need to sample $h(t)$ and $b_{T'}(t)$ first. From the Sampling Theorem, we know that for a bandlimited signal $X(f)$, if samples of $x(t)$ are taken 1/2W apart, then $x(t)$ can be completely recovered from the samples, where W is the bandwidth of the signal $x(t)$. How fast should we sample $h(t)$? $h(t)$ is assumed to be bandlimited because of the sinc function. Observing (4.1), we see that in frequency domain, $H(f)$ is the convolution of a chirp function and a window. Therefore, the bandwidth of $h(t)$ is infinite and there would be an infinite number of computations for $d_{T'}(t)$.

---

[1] In Mathematica, error function is defined as Erf(z) = $\frac{2}{\sqrt{\pi}}\int_0^z e^{-t^2}dt$.

At present time, a simplified $h(t)$ is used for (4.2). Let $\tilde{h}(t)$ denote the simplified version of $h(t)$, which is the periodic version of the $\frac{sin \pi x}{\pi x}$-shaped transform of a window. Then the question is how many samples we should use to digitize the periodic version of the sinc function. In the algorithm for generating the range profiles, the reflectivity of a target is digitized into 256 complex values. It is zero padded to 512 samples before taking the Fourier transform. Since the Fourier transform of $\tilde{h}(t)$ is an ideal lowpass filter with height 1, to avoid the aliasing, there should be at least 512 samples for the filter. Among those samples, 2L-1 are 1s. Hence, the sampled version of the pulse function of the periodic version of the sinc function is [11]

$$\begin{aligned} \tilde{h}(k) = & \frac{1}{512} \sum_{l=-L+1}^{L-1} e^{jkl2\pi/512} && (4.11) \\ = & \frac{1}{512} \frac{\sin(\frac{k2\pi}{512}(L-1/2))}{\sin(\frac{k2\pi}{1024})}, && (4.12) \end{aligned}$$

where L is number of samples on the periodic version of the sinc function. We want to solve for L. $\frac{k}{512}(2L-1) = 1$; if we assume k = 8, then L $\approx$ 32. In the simulation, 32 samples are used for the periodic version of the sinc function.

4.2 Validation of the Model

Validation of the model has two aspects. The first one is the validation of the range profile model. The second aspect is the validation of the template.

Validating the range profile model involves verifying that (3.6) is accurate enough to produce a predicted HRR signature so that a correct discrimination on the target type can be made. Assume a perfect template is available for this part of the validation. Integrated square-error is used as a measure of distance. A suggested procedure is to find the distance between the predicted HRR signature and the true HRR signature plus noise. In one extreme, that is the current model is perfect, the distance between the two signatures is a Chi-square distributed random variable with $2N$ degree of freedom if the additive noise in the true HRR signature is sampled to yield N independent complex random variables. If the distance between the two signatures is very large, there are two types of change that can be made. The first type is varying the terms in (3.6). A different $f(\cdot)$ can be chosen; a more accurate model can be used for the random phase; instead of 0 and 1, the illumination mask can take values in a larger set of numbers. If all these changes would not make the distance between the signatures close enough, a second type of change could be used: taking into account the electromagnetic interaction between patches. This change would increase the computational complexity of the algorithm because solving some maxwell equations may be required.

For validating the template, we assume that the current model is perfect. The objective here is to adequately model the target so that the predicted HRR signature generated using (3.6) is close to the true HRR signature. Euclidean distance is used as a measure of the closeness. The underlying assumption is that more patches will provide a better predicted HRR signature. One way of doing the validation is to find an initial template. Compute the distance between the true HRR signature with noise and the predicted HRR signature. If the distance is not small enough to find the right target type from the rest, then we can either increase the number of patches in the template or we can vary the location of the patches.

In Fig. 4.1, each circle, C_i, represents a set of predicted HRR signatures generated with i patches that are placed at different locations. X_i is a point on the set of C_i which corresponds to a predicted signature that is closest to the true signature.

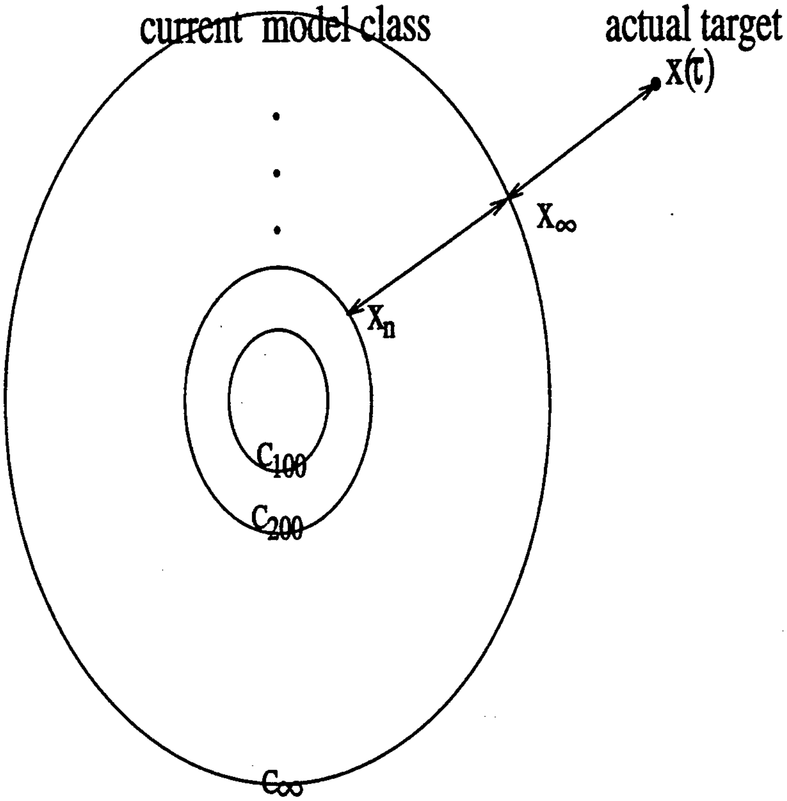


Figure 4.1: Error between the true HRR signature and predicted HRR signature

X_∞ corresponds to a perfect template for the current model. Let X denote the true HRR signature. The distance between X and X_∞ is the distance described in validating the range profile model case. The distance between X_∞ and X_n is the distance described in the validating the template case. The distance used here is the Euclidean distance. Mathematically,

$$C_\infty = \lim_{n \to \infty} C_n,$$

$$X_i = \arg\min_{X' \in C_i} d(X, X'),$$

and the distance between a point and a set is

$$d(X, C_i) = \min_{X' \in C_i} d(X, X').$$

By the triangle inequality,

$$d(X, C_n) \leq d(X, X_\infty) + d(X_\infty, C_n). \tag{4.13}$$

In (4.13), the first term on the right hand side corresponds to the modeling error. The second term corresponds to the template error. And the left hand side corresponds to the total error in the algorithm. From (4.13), we can conclude that decreases the distance in either terms which corresponds to better template and better model will yield a better estimation of the target type.

5. Results

The algorithm introduced in Chapter 3 for generating the range profiles of a target using a shape model having noninteracting surface patches is implemented on a SUN Sparc station IPX. In this chapter, we will first show some resulting range profiles. Then we will discuss the effect of using different parameters. A series of range profiles for the target with various orientations is also generated.

5.1 The Target and the Template

The target used for writing and testing our programs is a facet model of the fighter plane X-29 provided with a the Silicon Graphics demonstration. Fig. 5.1, Fig. 5.2, and Fig. 5.3 are the top view, bottom view, and side view of the X-29, respectively.

The template we used for the faceted model of the X-29 is stored as a list of position vectors and directional normal vectors. There are a total of 573 facets for the X-29 plane. The data set of the template is composed in such a way that the directional normal vectors are followed by the position vectors. The position vectors are defined by the x, y, z values of the corners of the facets. The directional normal vectors are defined by the unit normal vectors located at the corners of the facets and perpendicular to the corresponding areas. The data set begins with the total number of facets for the target. Then it is followed by $3 \cdot 2 \cdot 4 \cdot$ (number of facets) real numbers, where $3 \cdot 2$ corresponds to the position vector and the directional

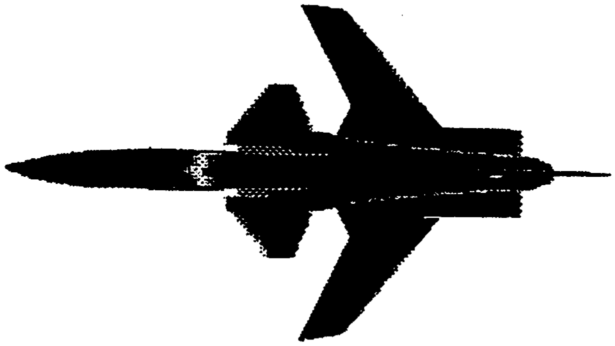


Figure 5.1: A faceted model of X-29 with 573 patches displayed on a Silicon Graphics machine, top view.

normal vector, 4 corresponds to the 4 corners of the quadrilateral. All the data are stored in the binary form. For example, after converting from binary form to decimal form, a data set may have the form of

573

unit normal vector x of patch 1 *corner* 1

unit normal vector y of patch 1 *corner* 1

unit normal vector z of patch 1 *corner* 1

position vector x of patch 1 *corner* 1

position vector y of patch 1 *corner* 1

position vector z of patch 1 *corner* 1

unit normal vector x of patch 1 *corner* 2

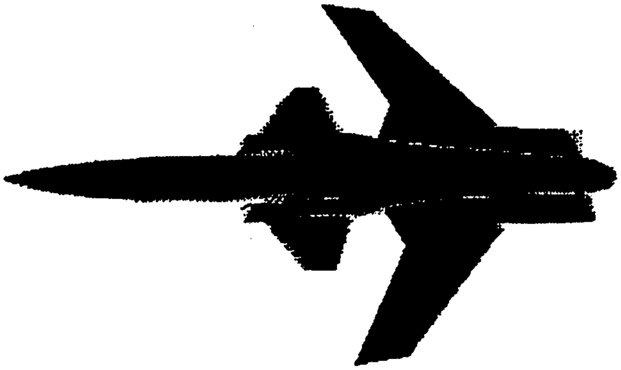


Figure 5.2: A faceted model of X-29 with 573 patches displayed on a Silicon Graphics machine, bottom view.

unit normal vector x of patch 573 *corner* 4

unit normal vector y of patch 573 *corner* 4

unit normal vector z of patch 573 *corner* 4

position vector x of patch 573 *corner* 4

position vector y of patch 573 *corner* 4

position vector z of patch 573 *corner* 4

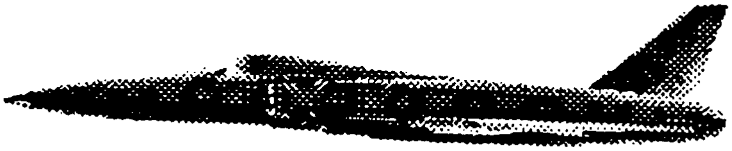


Figure 5.3: A faceted model of X-29 with 573 patches displayed on a Silicon Graphics machine, side view.

5.2 Implementation and the Results

In this section, we will give a general description about how we implement the algorithm and show a list of range profiles. Along the way, the data structure we used will be also introduced.

The implementation of the algorithm is divided into two major parts: making the illumination mask and generating the range profile. The illumination mask has the hash table data structure as shown in Fig. 5.4.

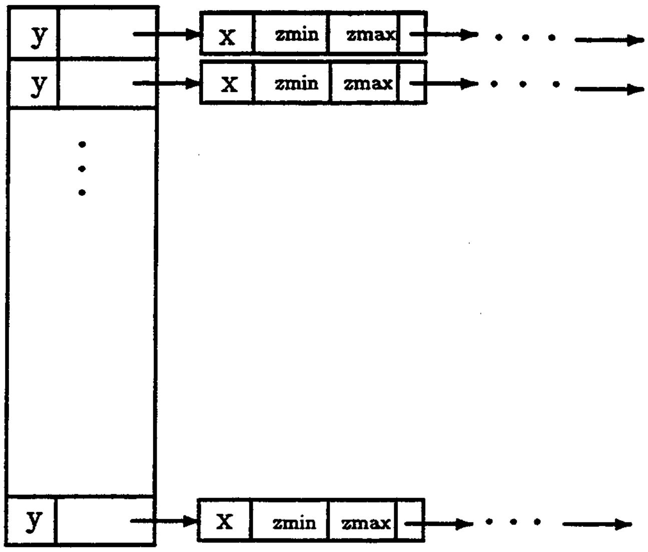


Figure 5.4: Hashing data organization for the shading mask. The main structure is an array. Each entry of the array is a structured component containing two fields: a real number and a pointer points to a linked list. The linked list has dynamic storage space.

Each element of the table has the structure of

```
typedef struct {
    float        y;
    stptr        b;
    } msk;
```

where stptr is a dynamically sized linked list. Each node of the list has the structure

```
typedef struct strip {
        float       x;
        float       zmin, zmax;
   struct strip     *next;
        } binptr;
```

The range profile has a simpler data structure which is only an array of complex numbers.

The data set of the X-29 template is read into an array of real numbers called pdata. The minimum and maximum x values of the original data set are found. The template is then scaled up proportionally in x, y, z such that the fuselage length of the plane is 10 meters. Then the illumination mask and the range profile are computed according to the algorithms described in Chapter 3. Below, we will show several groups of range profiles that are generated using different parameters. For all the groups, the range profiles are calculated according to (3.6). The displayed range profiles are the results from convolving the range profiles from (3.6) with the modified sinc function whose impulse response is given as (4.12) in Chapter 4.

Fig. 5.5 is a range profile for the X-29 plane. It is generated by letting b in (3.6) equal 1. There is only one uniform random phase for all patches, and (3.11) is used for $f(\cdot)$. Fig. 5.5 is the range profile of the plane in the tail aspect. We see that the tail wing and the tail provide large returns which is shown as the first peak. The second peak is from the back wings. The cockpit effect results the third peak. We also tested the program using different orientations. Andersh, et. al. [10] compared

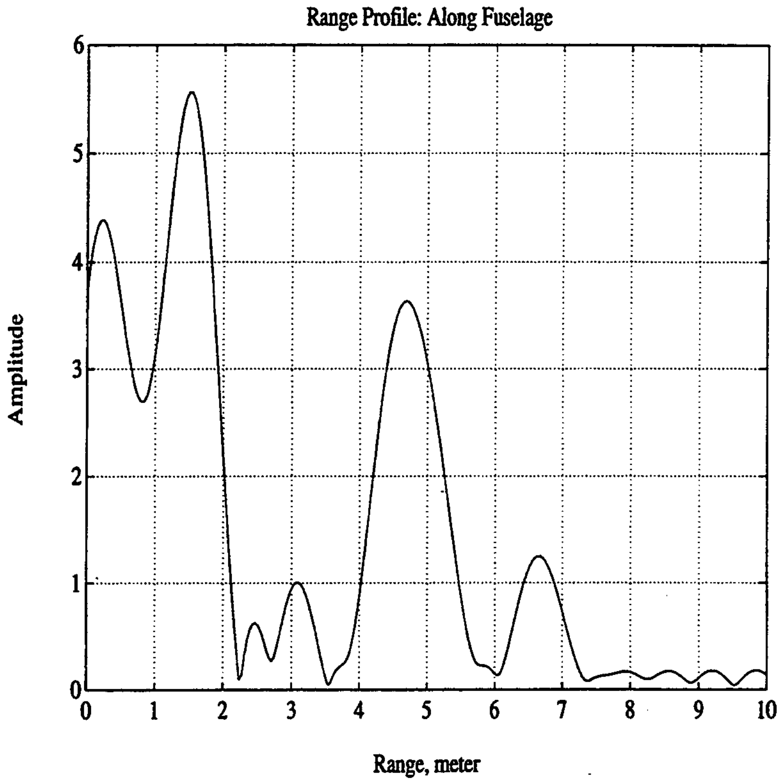


Figure 5.5: Range profiles of X-29. Tail Aspect

their synthesized range profile with measured data to demonstrate the accuracy of the prediction. However, for our algorithm, currently there is no measured data available for us to verify the synthesized range profile.

In the implementation, there are some parameters that must be chosen. Our next step is to check the effect of varying these parameters. The first group of range profiles we show is generated by varying the way of defining a shaded patch. As before, in (3.6), b is assumed to be 1. There is only one random phase for all the patches. (3.11) is used for $f(\cdot)$. The target is illuminated from the side. In this group, we will test the effect of varying the definition of a shaded patch. Three range profiles are generated using different definitions for a shaded patch. First, we define a patch to be shaded if more than one corner is shaded. Second, we define a patch to be shaded if more than two corners are shaded. Third, we define a patch to be shaded if more than three corners are shaded (i.e. a patch is not shaded unless all corners are shaded). Fig. 5.6 shows the range profiles. We observe that among the three range profiles, the magnitude of the reflectivity is largest when the third definition is used, and is smallest when the first definition is used.

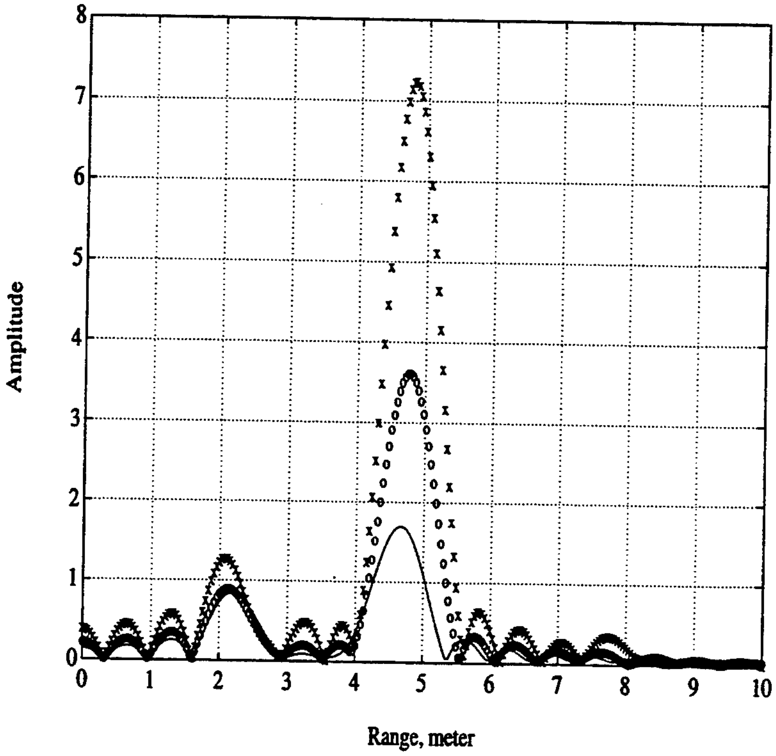


Figure 5.6: Range profiles for the same template with different definition of shaded patch. The crosses are the reflectivity for definition three. The circles are the reflectivity for definition two. The line is the reflectivity for definition one.

The second group of range profiles is generated to study the effect of different ways of modeling radar cross section. All the parameters are set to equal those in group one, and the second definition for a shaded patch from group one is used. Two sets of range profiles are generated to test the effect of different modeling. The first range profile is generated with $f(\cdot)$ equal to (3.11), the second range profile is generated with $f(\cdot)$ equal to (3.10). Fig. 5.7 shows the results. The circles are the reflectivity of the template if the antenna pattern is used. The line is the reflectivity of the template if the squared cosine relation is used. The conclusion is that different ways of modeling radar cross section do change the range profile.

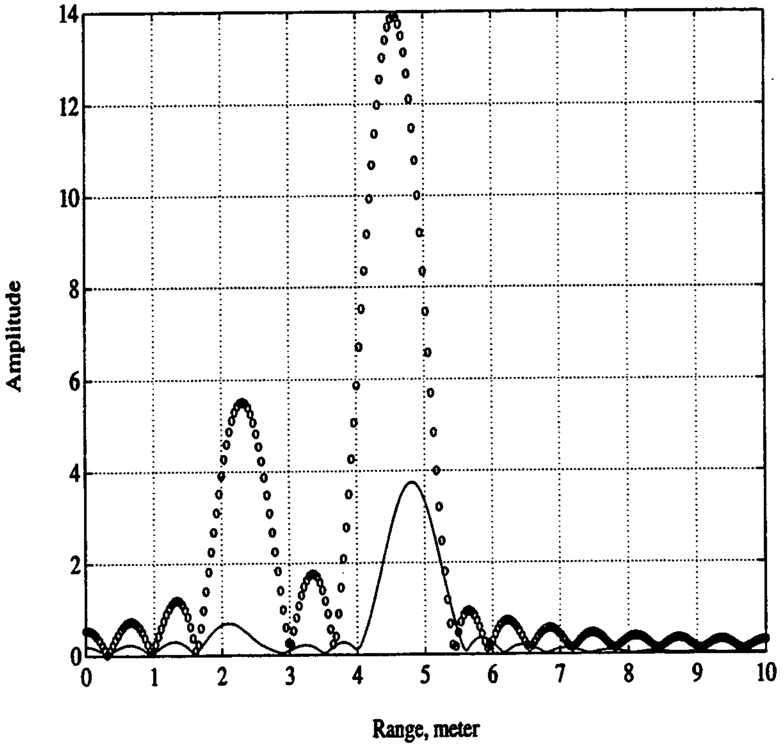


Figure 5.7: Range profiles for the same template with different ways of modeling the radar cross section. The circles are the reflectivity of the template if the antenna pattern is used. The line is the reflectivity of the template if the squared cosine relation is used.

The third group of the range profiles is generated by using the second definition from group one, one random phase for all patches, modeling the target cross section by the antenna pattern, and varying the orientation of the template (i.e. varying the yaw angle). Fig. 5.8 shows 20 sets of range profiles for the same target with

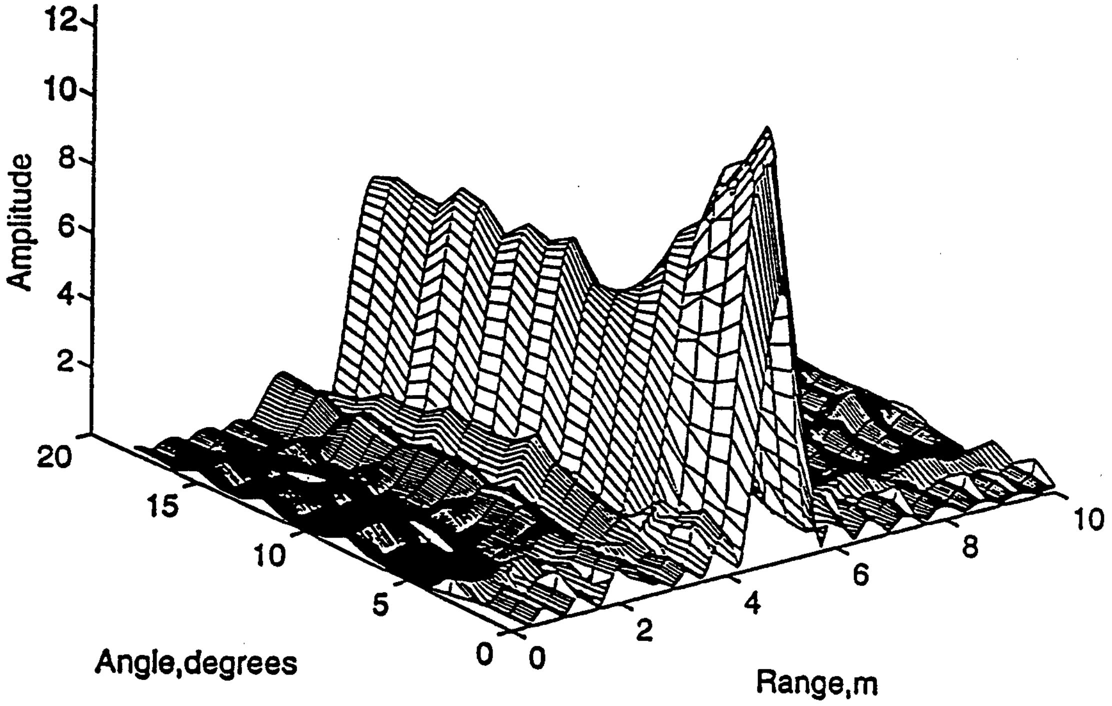


Figure 5.8: Range profiles for the same target with different orientations. There are no shifting and rotations in roll and pitch. The adjacent range profiles differs 1° in yaw. There is only one random phase for all patches.

different orientations. These range profiles correspond to rotating the target about the z-axis in 1° step for 20 degrees. There is no shifting in either of the axes; and there are no rotations about the x-axis and y-axis. The rotations of the template are reflected as skewing of the peaks of the corresponding range profiles.

The last group of the range profiles is generated to study the use of random phases in our model. All parameters are exactly the same as group three except that in this group there is a different random phase for each patch. This means that we have no knowledge about the phase shift between the reference point and the patch. This group of the range profiles makes a contrast to the third group in the sense of modeling the phase. Assigning different random phases to each patch and assigning one random phase to all patches are two extremes. The real situation lies somewhere in between and it is unknown at present time.

5.3 Summary

By studying various range profiles, we conclude that the different definitions of a shaded patch and the different ways of modeling the radar cross section only affect the magnitude of the reflectivity. For the random phase, it seems that assigning one random phase to all patches in the template is a more realistic thing to do. However, the modeling of the random phase is a subject should be studied in the future.

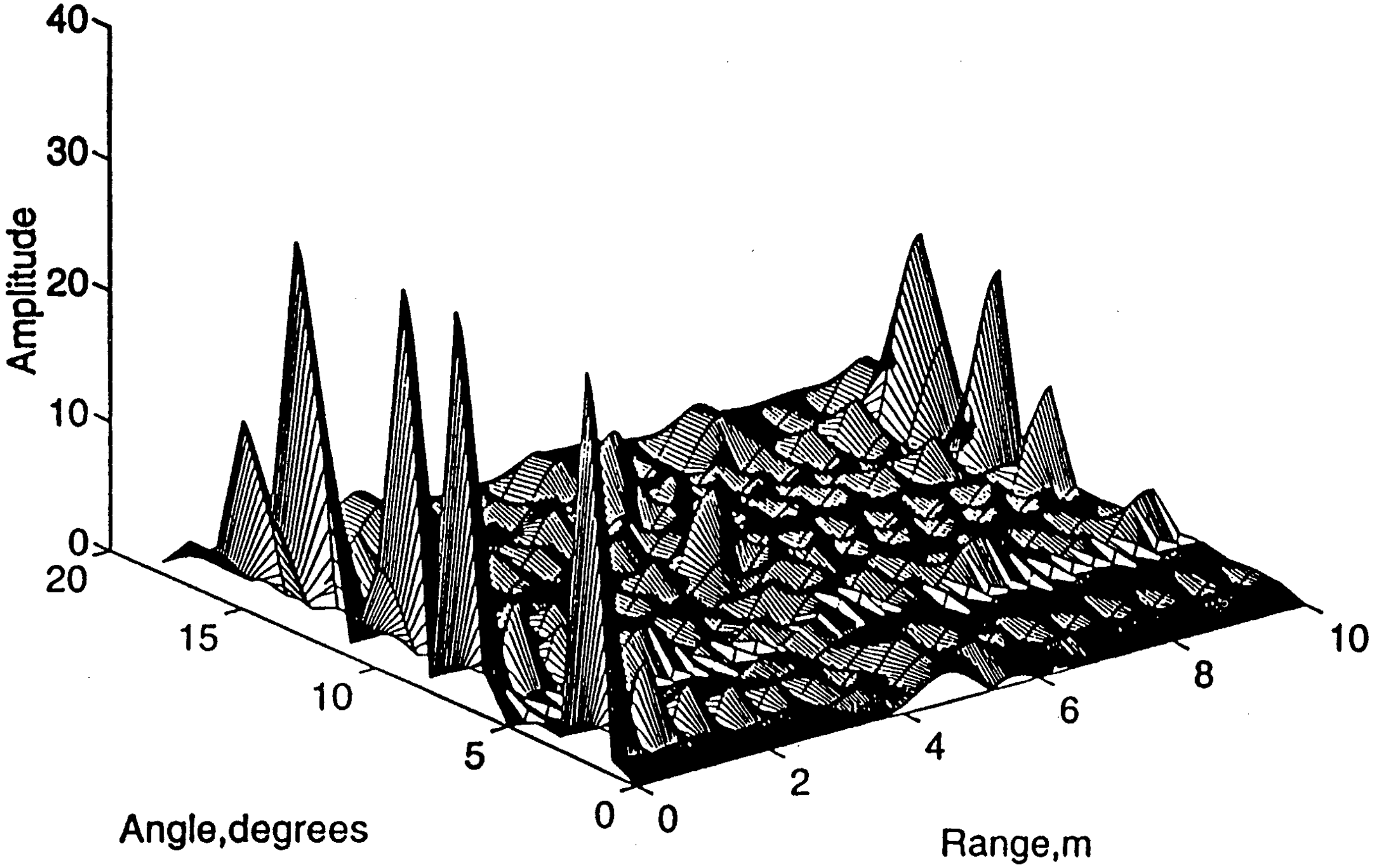


Figure 5.9: Range profiles for the same target with different orientations. There are no shifting and rotations in roll and pitch. The adjacent range profiles differs 1° in yaw. There are different random phases for each patch.

6. Conclusions and Future Work

An automatic target recognition system should be able to decide the number of targets in the scene, to classify the targets, and to track the targets. In order to achieve the target identification using IBS, range profile prediction is important. An algorithm is introduced in this thesis to synthesize the range profile of a target. A model for the range profile of a target is proposed. Based on the model, we divide the task of predicting the range profiles into two parts: generating the illumination mask and computing the reflectivity. The objective of this thesis is to synthesize the range profile of target for use in automatic target recognition. The range profile of the X-29 airplane is generated. There are some aspects of the algorithm that can be altered to improve the performance of the algorithm.

First, we believe that the result can be improved if a different $f(\cdot)$ is chosen for the range profile model. In this project, the facet model of a target is used. For the X-29, the target is partitioned into 573 quadrilateral patches. The four corners of the quadrilateral are assumed to be non-coplanar. We divide the quadrilateral into four smaller parallelograms as shown in Fig. 3.4. In our simulation, when (3.12) is used for $f(\cdot)$, it corresponds to the antenna pattern of a circular aperture centered at each corner of the quadrilateral, with area equal to the area of the parallelogram. An improvement is to find the $f(\cdot)$ which corresponds to the antenna pattern of the parallelograms. Also, as we stated at the beginning of this thesis, this model does not account for the inter-patch electromagnetic effects. An incorporation of these effects may have a significant impact on the resulting range profile.

A second issue is the choice of whether to use a random phase in our model. We know that assigning one random phase to all patches and assigning different random phases to all patches are two extreme cases. Where the real situation stands is a topic that must be studied in the future. By observing Fig. 5.8 and Fig. 5.9, we think the real situation should lie more toward the case of assigning one random phase to all patches.

A third issue is that the validation of the model and the template suggested in Section 4.2 should be carried out to foresee the possible improvement that can be made.

The algorithm is tested on the X-29 provided by the Silicon Graphics demonstration. It has been proven successful in this target. one current limitation of this implementation is that no actual measured data are available to compare to the simulated data. Data from a variety of targets should be collected and compared to the output of our range profile prediction.

7. APPENDIX

```
/* ***********************************************************
   * following procedures pad 0s to the original range profile;*
   * take fft, multiply by a window [-16,16]; and then take    *
   * a ifft.         *
   ********************************************************** */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define range_num 256
#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr

typedef struct {
   float real;
   float imag;
  } cmplx;

void getdata(reflect,data,n)
cmplx reflect[range_num];
float data[(range_num)*4];
int n;

{
  int i;

  for (i=1;i<=n/2;i++)
    {
     data[i*2-1]=reflect[i-1].real;
     data[i*2] = reflect[i-1].imag;
    }
  for (i=n+1;i<=n*2;i++)
    data[i]=0;
}

void four1(data,nn,isign)
float data[512];
int nn,isign;
{
        int n,mmax,m,j,istep,i;
```

```
double wtemp,wr,wpr,wpi,wi,theta;
float tempr,tempi;

n=nn << 1;
j=1;
for (i=1;i<n;i+=2) {
        if (j > i) {
                SWAP(data[j],data[i]);
                SWAP(data[j+1],data[i+1]);
        }
        m=n >> 1;
        while (m >= 2 && j > m) {
                j -= m;
                m >>= 1;
        }
        j += m;
   }
mmax=2;
while (n > mmax) {
        istep=2*mmax;
        theta=6.28318530717959/(isign*mmax);
        wtemp=sin(0.5*theta);
        wpr = -2.0*wtemp*wtemp;
        wpi=sin(theta);
        wr=1.0;
        wi=0.0;
        for (m=1;m<mmax;m+=2) {
                for (i=m;i<=n;i+=istep) {
                        j=i+mmax;
                        tempr=wr*data[j]-wi*data[j+1];
                        tempi=wr*data[j+1]+wi*data[j];
                        data[j]=data[i]-tempr;
                        data[j+1]=data[i+1]-tempi;
                        data[i] += tempr;
                        data[i+1] += tempi;
                }
                wr=(wtemp=wr)*wpr-wi*wpi+wr;
                wi=wi*wpr+wtemp*wpi+wi;
        }
        mmax=istep;
}
```

```
}

#undef SWAP

void cheat(data)
float data[1024];

{
  int l;
  int i;

  l = 16;
  for (i=l;i<=(512-l);i++)
    data[i*2] = data[i*2+1] = 0;
}

void covsinc(reflect)
cmplx reflect[range_num];
{
float refl[1024];
int n,i;
int isign;
float mag;

n = range_num * 2;
getdata(reflect,refl,n);
four1(refl,n,1);
cheat(refl);
four1(refl,n,-1);
for (i = 1;i<=256;i++)
   {
    reflect[i-1].real = refl[2*i-1];
    reflect[i-1].imag = refl[2*i];
   }
}
```

```
/* *************************************************************
   * This subroutine will create up to 100 different file names *
   * for storing the range profiles. *
   ************************************************************* */
#include <stdio.h>

void inc(filename, digit)
char filename[20];
int digit;

{
  printf("digit = %d\n",digit);
  if ((digit > 7) || (digit < 2))
   {
    printf("digit = %d\n",digit);
    printf("something is wrong \n");
    exit(1);
   }
 else {
  if (filename[digit] == '9')
    {
     filename[digit] = '0';
     inc(filename,digit-1);
    }
   else
    filename[digit]++;
 }
}
```

```
/* ****************************************************************
   * readdata reads in the template from a data file              *
   ************************************************************** */

#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>

void readdata(pp,sizep,argv)
float **pp;
long *sizep;
char argv[20];
{
        int data_file;
        long magic,
             zero;

/* check if file name specified */
     /*   if (argc < 2) {
                printf("usage: display filename[.bin]\n");
                exit(0);
        } */

/* open binary data file */
        if ((data_file = open(argv,O_RDONLY,0)) == -1) {
                /* file does not exist, add .bin extension */
                if ((data_file = open(strncat(argv[1],".bin",4),
                    O_RDONLY,0)) == -1){
                        /* file.bin does not exist either */
                  argv[strlen(argv[1])-4]='\0';
                  printf("%s: No such file or directory\n",argv[1]);
                  exit(0);
                }
        }

/* read magic num, 4 bytes */
        read(data_file,&magic,4);

/* read object size and zero field */
        read(data_file,sizep,4);
        read(data_file,&zero,4);
/*        printf("object size: %d polygons.\n",(*sizep)/4);*/

/* allocate space:
   <size> vertices, 4 bytes per coordinate, 3 coordinates
   for each point,  and a normal (also 4x3). */
        *pp = (float *) malloc((*sizep)*4*3*2);

/* read data */
        read(data_file,*pp,*sizep*4*3*2);
        close(data_file);
}
```

```
/* ***************************************************************
   * This program accomplishes following tasks according to the   *
   * list sequence:                                             *
   * 1. shift and rotate to the appropriate coordinate            *
   * 2. generate a mask which is a structure array, each element  *
   *    of the array is a linked list containing the patches that *
   *    are not shaded by the patches in front of them            *
   * 3. calculate the area and compute the projected area in the  *
   *    direction of the signal for the patches that are not      *
   *    shaded.               *
   * 4. generate a uniformly distributed random phase from -pi to *
   *    pi.          *
   * 5. multiply the random phase and the projected area          *
   * 6. sum up the reflectivity accroding to tau            *
   * 7. take the FFT of the reflectivity, multiply by a low pass  *
   *    filter, then take the IFFT of the result   *
   ************************************************************** */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "readdata.c"
#include "frange.c"                              /*perform FFT & IFFT*/
#include "namecreat.c"

#define RAND_MAX 32767
#define pi 3.141592654
#define range_num 256                   /*tau range*/
#define X 3
#define Y 4
#define Z 5
#define Xn 0
#define Yn 1
#define Zn 2

#define resln 100 /*y range resolution*/

typedef struct {
   float x,y,z;        /*position*/
   float xn,yn,zn;   /*normal*/
   } CN;                                    /*patch corner structure*/
```

```
/*typedef struct {
   float real;
   float imag;
  } cmplx;     */      /*complex number type*/

typedef struct  {
   float x1,y1,z1;
   float x2,y2,z2;
   } bndptr; /*corner boundary type*/

typedef struct strip{
   float x;                           /* dist from radar */
   float zmin;      /* vertical range */
   float zmax;
   struct strip *next;      /* next strip */
   } binptr;

typedef binptr *stptr;

typedef struct  {
   float  y;
   stptr  b;        /*list of patches for y  */
   } msk;

typedef enum { false=0, true=1 } boolean;


/* ============================================================= */
void init(arr,dim)

/* init will initialize a real array with dimension dim*/
/* ************************************************************* */
float *arr;
int dim;
{
  int i;

    for (i = 0;i < dim;i++)
       arr[i]=0;
}/*end of init*/
```

```
/* ============================================================= */



/* ============================================================= */
void cmpinit(arr,dim)

/*cmpinit will initialize a complex array with dimension dim*/
/* ************************************************************* */
cmplx *arr;
int dim;
{
  int i;

  for (i =0;i<dim;i++)
      arr[i].real = arr[i].imag = 0;
} /*end of cmpinit*/
/* ============================================================= */



/* ============================================================= */
void rmatrix(R,r1)

/*This subroutine computes the rotating matrix
  Input R: 3x1, contains three desired angle for x,y,z rotating
  Output r1: 3x3, the matrix needs to be multiplied by the old  */
/* ************************************************************* */

float R[3];
float r1[3][3];
{

 r1[0][0]=cos(R[2])*cos(R[1]);
 r1[0][1]=cos(R[1])*sin(R[2]);
 r1[0][2]=-sin(R[1]);
 r1[1][0]=sin(R[0])*sin(R[1])*cos(R[2])-cos(R[0])*sin(R[2]);
 r1[1][1]=sin(R[0])*sin(R[1])*sin(R[2])+cos(R[0])*cos(R[2]);
 r1[1][2]=sin(R[0])*cos(R[1]);
 r1[2][0]=cos(R[0])*sin(R[1])*cos(R[2])+sin(R[0])*sin(R[2]);
 r1[2][1]=cos(R[0])*sin(R[1])*sin(R[2])-sin(R[0])*cos(R[2]);
 r1[2][2]=cos(R[0])*cos(R[1]);

} /*end of rmatrix*/
/* ============================================================= */



/* ============================================================= */
void Rota(r,V)

/* This subroutine rotates the XYZ from current coordinate
   to the new coordinates.
   r:  input/output.  It has the current  XYZ;
       and will be replaced by the new values.
   V:  a 3x3 transformation matrix.              */
/* ********************************************************* */

float r[3][3],V[3];
{
  float W[3];
  int i,j;

  for (i = 0; i<3; i++)
   W[i]=0;

  for (i = 0; i<3; i++)
    for (j = 0; j<3; j++)
      W[j] = W[j] + r[j][i]*V[i];      /* matrix r times vector v */

   for (i = 0; i<3; i++)
     V[i]=W[i];
}  /* end of Rota */
/* ============================================================= */



/* ============================================================= */
void area(corner,A,ind,w)

/* This subroutine will calculate the areas of the given patch.
   A1: area bounded by P1P2,P2P3
   A2: area bounded by P2P3,P3P4
   A3: area bounded by P3P4,P4P1
   A4: area bounded by P4P1,P1P2                    */
/* ********************************************************* */

CN corner[4];
float A[4];
int ind[2292],w;
{
  float DP[4][3];
  float t1,t2,t3;
  int i,j,k;
  int u,v;

  for (i=0;i<4;i++)
    {
     DP[i][0] = corner[(i+1)%4].x - corner[i%4].x;
     DP[i][1] = corner[(i+1)%4].y - corner[i%4].y;
     DP[i][2] = corner[(i+1)%4].z - corner[i%4].z;
    }


 /* if the patch is shaded, ind = 1, o/w ind = 0 */
   for (i=0;i<4;i++)
     if (ind[w*4+i] == 0)
      {
       u = (i+1)%4;
       v = i%4;
       t1 = DP[u][1]*DP[v][2] - DP[v][1]*DP[u][2];
       t2 = DP[u][0]*DP[v][2] - DP[v][0]*DP[u][2];
       t3 = DP[u][0]*DP[v][1] - DP[v][0]*DP[u][1];
       A[i] = 0.25 * sqrt(t1*t1+t2*t2+t3*t3);
      }
     else   /*the area is shaded*/
       A[i] = 0;
  } /* end of area */
/* ================================================================ */



/* ================================================================ */
```

```
void projarea(corner,A)

/* This procedure reads in the areas of one patch,
   computes and returns the projected areas.
   This is the sqaured-cosin dependence method.      */
/* ************************************************************ */

CN corner[4];
float A[4];
{
int i;

 for (i=0;i<4;i++)
   /* if (corner[i].xn >= 0)
      A[i]=0;
    else*/
      A[i] = -corner[i].xn * A[i];
} /* end of projected area */
/* ============================================================= */



/* ============================================================= */
void J_area(corner,A,lambda)

/* This procedure is an implementation of the cross
   section using antenna pattern.
   A: input/output.  it is the original area and will
      be replaced by the resulting value
   lambda: is the wavelenth of the carrier.       */
/* ************************************************************* */

CN corner[4];
float A[4];
float lambda;
{
int i;
float r;
float p;
float param;
```

```
for (i=0;i<4;i++)
  {
   if (A[i] != 0)
     {
      r = sqrt(A[i]/pi);
      p = 2*pi*sin(acos(-corner[i].xn))*r/lambda;
      if (p == 0)
       A[i] = 0.5; /*special case of the antenna function for 0/0 */
      else
         A[i] = A[i] * ((float)(j1(p)) / p);
     }
  }
}/*end of J_area*/
/* ============================================================ */



/* ============================================================ */
void randpha(ph,n)

/* This procedure generates a random number which is uniformly
   distributed from -pi to pi.  Then sine and cosin of the
   number are calulated for the random phase.             */
/* ********************************************************** */

cmplx *ph;
int n;
{
  int i,rn;
  float nrn;

  srand(1);

  for (i=0;i<n/4;i++)
    {
   /*set random # generator to a starting pt particular to the seed*/
     rn = rand();

   /*a uniformly distributed random phase in the range [-pi,pi]*/
     nrn = ((float) rn)/RAND_MAX * 2 * pi - pi;
     /* real part of the phase */
     ph[i].real = ((float)cos(nrn));

     /* imaginary part of the phase */
     ph[i].imag = ((float)sin(nrn));
    }
} /* end of randpha */
/* ============================================================= */



/* ============================================================= */
void sum_reflect(corner,a,phase,xi,xa,reflt)

/* This procedure sums up the reflectance of the target according
   to the range.
   input: corner, a, phase, xi, xa
     a: area of the region.
     xi, xa: minimum and maximum of the target in x
             direction.  xa - xi = 10 m.
   output: reflt, reflectivity of the target, complex
           valued. */
/* *********************************************************** */

CN corner[4];
float xa,xi;
float a[4];
cmplx reflt[range_num],phase;
{
 int xstep;
 int i;
 int sl;
 float step;

 step = (xa-xi)/(range_num-1);
 for (i=0;i<4;i++)
  {
    sl = (int)(ceil((float)((corner[i].x-xi-0.000001)/step)));
    reflt[sl].real = reflt[sl].real+a[i]*phase.real;
    reflt[sl].imag = reflt[sl].imag+a[i]*phase.imag;
  }
```

```
} /*end of sum_reflect */
/* ============================================================ */



/* ============================================================ */
void InitMask(mask,ygmin,delta)

/* This procedure initializes the shading mask.
   input: ygmin, delta
     ygmin:  minimum value of the target in y direction.
     delta:  width of the columnwise patitioned
       grid.
   output: mask, a hash table  */
/* ********************************************************* */

msk *mask;
float ygmin,delta;
{
float y;
int i;

   y = ygmin + delta/2;
   for (i=0;i<resln;i++)
      {
         mask[i].y = y;
         mask[i].b = NULL;
         y += delta;
      }
}  /* InitMask */
/* ============================================================ */



/* ============================================================ */
void GetBounds(data,corner)

/* fetches perimeter of patch stored in slope-intercept form.
   assumes perimeter is defined as lines from 1->2, 2->3,
   3->4 and 4->1.
   Input: p, a pointer to the variable space
   Output: e[], a 4-member array of pointers that define a patch.*/
/* ************************************************************ */

float *data;
bndptr corner[4];
{
int i;
float *qdata;

   qdata = data;
   for (i=0;i<3;i++)
      {
         corner[i].x1 = qdata[X+i*6];
         corner[i].x2 = qdata[X+(i+1)*6];
         corner[i].y1 = qdata[Y+i*6];
         corner[i].y2 = qdata[Y+(i+1)*6];
         corner[i].z1 = qdata[Z+i*6];
         corner[i].z2 = qdata[Z+(i+1)*6];
      }
   corner[i].x1 = qdata[X+18];
   corner[i].x2 = qdata[X];
   corner[i].y1 = qdata[Y+18];
   corner[i].y2 = qdata[Y];
   corner[i].z1 = qdata[Z+18];
   corner[i].z2 = qdata[Z];
}  /* GetBounds */


/* ============================================================ */
int GetBin(y,min,max,delta)

/* computes the appropriate bin number corresponding to y
   Input: y, y value that the bin number corresponding to
          min, minimum y value of all the patches
          max, maximum y value of all the  patches
          delta, distance of y from bin n+1 to bin n
   Output: bin number                                         */
/* ************************************************************ */

float y,min,max;
float delta;
{
 int test;
 float temp2;

   if (y == max)
    return(resln-1);
   else
    {
     temp2 = (y-min)/delta;
     test = (floor)(temp2);
     return (test);
    }
}  /* GetBin */
/* ============================================================= */



/* ============================================================= */
boolean InRange(biny,corner1y,corner2y)

/* check if the line connect corner1 and corner2 is in the bin
   Input: biny, y value of the current bin
        corner1y, y value of the corner that is checking
        corner2y, y value of the corner that is next to the corner1
   Output: 1 or 0.  1 means that the line is in the current bin,
           0 otherwise.                                            */
/* *************************************************************** */

float biny,corner1y,corner2y;
{
   if ((biny>=corner1y)&&(biny<=corner2y))
      return(true);
   else if ((biny<=corner1y)&&(biny>=corner2y))
        return(true);
   else return(false);
}     /* InRange */
/* ============================================================= */
/* =============================================================== */
void Assign(p,min,max,xx,nx)

/* assign the values to a node of type binptr */
/* ******************************************************* */

binptr *p,*nx;
float min,max,xx;
{
   p->zmin = min;
   p->zmax = max;
   p->x    = xx;
   p->next = nx;
}  /* Assign */
/* =============================================================== */



/* =============================================================== */
void maintain(lst,n,pn)

/* maintains the linked list of the bin such that for different x,
   no zs' overlap.
   lst: is the proper linked list containing x, zmin, and zmax.
   n:   is a pointer to the new node which is just inserted. It
        will be checked against the lst for shadowing.
   pn:  is a pointer points to the node previous to node n.    */
/* *************************************************************** */

binptr **lst,*n,*pn;

{
  binptr *beg,*bottom,*top,*cross,*pcross;
  binptr *morelst,*tpn,*GC;
  int flag1,flagi,flago;

  if (pn == NULL)
    flag1 = 1;
  if (n->next == NULL)
    flag1 = 3;
  if ((pn != NULL) && (n->next != NULL))
```

```
    flag1 = 2;

  morelst = (binptr *) malloc(sizeof(binptr));
  if (morelst == NULL){
    printf("not enough space to malloc for morelst\n");
    sleep(5);
   }
  Assign(morelst,n->zmin,n->zmax,n->x,NULL);

/* maintains for the part of the list before node n */
  beg = morelst;
  flago = 0;
  if ((flag1 == 2) || (flag1 == 3))
    {
     cross = *lst;
     pcross = NULL;
     while ((cross != n) && (flago == 0))
      {
       tpn=NULL;
       morelst = beg;
       while ((morelst != NULL) && (flago == 0))
        {
        if ((morelst->zmin<cross->zmin)&&(morelst->zmax>cross->zmax))
         {
          bottom = (binptr *) malloc(sizeof(binptr));
          top = (binptr *) malloc(sizeof(binptr));
          if ((top == NULL) || (bottom == NULL))
          printf("not enought space to malloc for top or bottom1\n");
          Assign(bottom,morelst->zmin,cross->zmin,morelst->x,top);
          Assign(top,cross->zmax,morelst->zmax,
                                 morelst->x,morelst->next);
          if (tpn != NULL)
              tpn->next = bottom;
          else
              beg = bottom;
          free(morelst);
          morelst=morelst->next;
          tpn = top;
          }
        else if ((morelst->zmin>=cross->zmin)&&
                                (morelst->zmax<=cross->zmax))
```

```
                {
                 if ((tpn == NULL) && (morelst->next == NULL))
                   flago = 1; /*the node added is complete shaded*/

                 else
                   {
                      if ((tpn == NULL) && (morelst->next != NULL))
  {
        beg = morelst->next;
free(morelst);
morelst = beg;
                         }
                      else if (tpn!=NULL)
                         {
                           tpn->next = morelst->next;
 free(morelst);
                           morelst = tpn->next;
                         }
                     }
                   }
        else if ((morelst->zmin>=cross->zmin)
                 &&(morelst->zmax>cross->zmax)
                 &&(morelst->zmin<cross->zmax))
           {
             morelst->zmin=cross->zmax;
             tpn = morelst;
             morelst = morelst->next;
           }
        else if ((morelst->zmin<cross->zmin)
   &&(morelst->zmax<=cross->zmax)
                 &&(morelst->zmax>cross->zmin))
          {
           morelst->zmax = cross->zmin;
           tpn = morelst;
           morelst = morelst->next;
          }
          else if ((morelst->zmax<=cross->zmin)||
                              (morelst->zmin>=cross->zmax))
          {
           tpn=morelst;
           morelst=morelst->next;
```

```
              }
             }          /*end of inner while*/
            cross = cross->next;
            pcross = cross;
        }               /*end of outer while*/
      }                 /*end of if*/
/* finishing maintain the part of the list before node n */

  if (flago == 1)       /*nothing to add in*/
    pn->next = n->next;

/* maintains the part of the list after node n */
  flagi = 0;
  if (((flag1 == 1) || (flag1 == 2)) && (flago != 1))
   {
    cross = beg;
    pcross = NULL;
    while ((cross != NULL) && (flagi == 0))
     {
      tpn = NULL;
      morelst = n->next;
      while ((morelst != NULL) && (flagi == 0))
       {
        if((morelst->zmin<cross->zmin)&&(morelst->zmax>cross->zmax))

          {
           bottom = (binptr *) malloc(sizeof(binptr));
           top = (binptr *) malloc(sizeof(binptr));
           if ((bottom == NULL) || (top == NULL))
               printf("not enough space for top or bottom2\n");
           Assign(bottom,morelst->zmin,cross->zmin,morelst->x,top);
           Assign(top,cross->zmax,morelst->zmax,morelst->x,
                  morelst->next);
           if (tpn != NULL)
             tpn->next = bottom;
            else
              n->next = bottom;
           free(morelst);
      morelst = top->next;
           tpn = top;
           }
          else if ((morelst->zmin>=cross->zmin)
&&(morelst->zmax<=cross->zmax))
                 {
                   if ((tpn == NULL) && (morelst->next == NULL))
                     flagi = 1;
                   if ((tpn == NULL) && (morelst->next != NULL))
                     {
                      n->next = morelst->next;
         free(morelst);
      morelst = n->next;
                      }

                    if (tpn != NULL)
                           {
                             tpn->next = morelst->next;
        free(morelst);
                             morelst = tpn->next;
                           }
                     }

             else if ((morelst->zmin>=cross->zmin) &&
                           (morelst->zmax>cross->zmax) &&
                                 (morelst->zmin<cross->zmax))
              {
                morelst->zmin=cross->zmax;
                tpn = morelst;
                morelst = morelst->next;
              }
           else if ((morelst->zmin<cross->zmin)
   &&(morelst->zmax<=cross->zmax)
                   &&(morelst->zmax>cross->zmin))
             {
              morelst->zmax = cross->zmin;
              tpn = morelst;
              morelst = morelst->next;
             }
            else if ((morelst->zmax<=cross->zmin)
                              ||(morelst->zmin>=cross->zmax))
              {
                tpn=morelst;
                morelst=morelst->next;
            }         /*end of inner while*/
         }
           pcross = cross;
           cross = cross->next;
      }               /*end of outer while*/
    }                 /*end of if*/


if ((flagi == 1) && (flago != 1)) /*everything after is shadded*/
   {
    if (pn == NULL)
      *lst = beg;
    else
      pn->next = beg;
    while (n != NULL)
     {
      GC = n->next;
      free(n);
      n = GC;
     }
   }
else if ((flagi != 1)&&(flago != 1))
   {
     if (pn == NULL)
      {
       *lst = beg;
       if (pcross==NULL)
         (*lst)->next = n->next;
       else
         pcross->next=n->next;
       }
     else
       {
        pn->next = beg;
        if (flag1 == 3)
           tpn->next = n->next;
        else
          pcross->next = n->next;
       }
    free(n);
  }
```

```
}/*end of maintain*/
/* =============================================================== */



/* =============================================================== */
float Interp(u,ur,wr)

/* This procedure checks the special cases for ur = 0 or wr = 0.
   If neither ones is zero, then u*wr/ur is computed.  Otherwise
   0 is returned.  */
/* ************************************************************* */

float u,ur,wr;

{
   if ((ur==0)||(wr==0))
      return(0);
   else
      return( u*wr/ur );
}  /* Interp */
/* =============================================================== */



/* =============================================================== */
void Insert(beg,n,pn)

/* This procedure inserts a node into a list pointed by beg
   in the increasing order of  the x values.  It also points
   pn to the node which is previous to n after it is inserted
   into the list.  If n points to the first node in the list,
   then pn points to NULL.         */
/* ************************************************************** */

binptr **beg,*n,**pn;

{
 binptr *tempbeg,*test,*ptest,*try;
 int flag;
```

```
   flag = 0;
   (*pn) = NULL;


   if (*beg == NULL)
    {
     *beg = n;
     (*beg)->next = NULL;
     tempbeg = NULL;
     flag = 1;
    }
   else
     tempbeg = *beg;

   while (tempbeg != NULL)
    {
    if (((tempbeg->x)>(n->x))&&(*pn == NULL))/*insert at beginning*/
       {
         *beg = n;
         n->next = tempbeg;
         flag = 1;
        }

    if (((tempbeg->x)>(n->x))&&(*pn != NULL))/*insert in the middle*/
        {
         (*pn)->next = n;
         n->next = tempbeg;
         flag = 1;
        }

     if (flag == 0)
        {
         *pn = tempbeg;
         tempbeg = tempbeg->next;
         }
     else
          tempbeg = NULL;
    }
    if (flag == 0)
          (*pn)->next=n;
} /*end of Insert*/
```

```
/* ============================================================== */



/* ============================================================= */
void Revise(lst)


binptr **lst;

{
 binptr *cross,*tail,*try;
 int ind;

 cross = *lst;
 tail = NULL;
 while (cross != NULL)
  {
   ind = 0;
   if (cross->zmin == cross->zmax)
     {
       ind = 1;
       if (tail == NULL)
          *lst = cross = (*lst)->next;
       else
         {
          tail->next=cross->next;
          cross = cross->next;
         }
       }

   else if (cross->next != NULL)
   {
    if ((cross->x==cross->next->x)&&(cross->zmin==cross->next->zmax))
        {
          ind = 1;
          cross->zmin = (cross->next)->zmin;
          cross->next = (cross->next)->next;
         }
     else if ((cross->x==cross->next->x)
  &&(cross->zmax==cross->next->zmin))
```

```
               {
                ind = 1;
                cross->zmax = (cross->next)->zmax;
                cross->next = (cross->next)->next;
                }
      }
      if (ind == 0)
        {
         tail = cross;
         cross = cross->next;
        }
     }
}/*end of revise*/
/* ============================================================== */



/* =============================================================== */
void Trace(corner,mask,delta,ygmin,ygmax)

/* This procedure computes zmax, zmin, and x values for the bins
   associated with each patch.
   corner:  position vectors of a quadrilateral patch.
   mask  :  shading mask of the template.
   delta :  width of the bin in the columnwise patitioned grid.
   ygmin,ygmax:  minimum and maximum value of the template in
                 y direction.     */
/* *************************************************************** */

bndptr corner[4];
msk *mask;
float delta,ygmin,ygmax;
{
float xx,yy,zz;
float dx,dy,dz;
float zlo,zhi;    /* min and max values of z for each bins */
float ylo,yhi;    /* min and max values of y for  the  given patch */
int i,j;
int first,last;  /* smallest and largest bin numbers for the patch*/
boolean flag;
binptr *beg,*pn,*try,*n;
```

```
int CONT;

   ylo = yhi = corner[0].y1;
   for (j=1;j<4;j++)
      {
         if (corner[j].y1 < ylo)
            ylo = corner[j].y1;
         if (corner[j].y1 > yhi)
            yhi = corner[j].y1;
      }
   first = GetBin(ylo,ygmin,ygmax,delta);
   if ((mask[first].y < ylo)&&(first<resln-1))
    first++;
   last = GetBin(yhi,ygmin,ygmax,delta);
   if ((mask[last].y > yhi)&&(last>0))
    last=last-1;
   for (i=first;i<=last;i++)
      {
         CONT=0;
         flag = false;
         for (j=0;j<4;j++)
           {
            if (InRange(mask[i].y,corner[j].y1,corner[j].y2) == 1)
               {  CONT=1;
                  dx = corner[j].x2 - corner[j].x1;
                  dy = corner[j].y2 - corner[j].y1;
                  dz = corner[j].z2 - corner[j].z1;
                  zz = Interp((mask[i]).y-corner[j].y1, dy,dz)+
                       corner[j].z1;

                  if (flag==false)
                     {
                        flag = true;
                        zlo = zhi = zz;
                     }
                  else
                     {
                        if (zz < zlo)
                          zlo = zz;
                        if (zz > zhi)
                          zhi = zz;
```

```
                      }
                     xx = Interp((mask[i]).y-corner[j].y1,dy,dx).
                          +corner[j].x1;
                 }
              }
      /* found the lo and hi */
        if (CONT==1)
         {
          beg = mask[i].b;
          n = (binptr *) malloc(sizeof(binptr));
          if (n == NULL)
            printf("not enough space for n to malloc\n ");
          Assign(n,zlo,zhi,xx,NULL);
          Insert(&beg,n,&pn);
          if (beg->next != NULL)
           {
            maintain(&beg,n,pn);
            /*Revise(&beg,acc,i);*/
           }
          mask[i].b = beg;
         }
        }  /*end of if*/
}  /* Trace */
/* =============================================================== */


/* =============================================================== */
void chk_shade(data,num,ymin,ymax,mask,ind)

/* This procedure decides whether the patch is shaded
   given the shading mask.
   input:  data, num, ymin, ymax, mask.
      data:  template in inertial frame
      ymin,ymax:  min and max y of the template.
      num:   number of patches in the template
   output:  ind
      ind:  is set to either 1 or 0.  1=not shaded         */
/* ************************************************************* */

float *data;
float ymin,ymax;
msk *mask;
int num;
int ind[2292];
{
int i,j;
float delta,*qdata;
binptr *front,*tail;
int tind,back,back2;
int count,binum,count2;
bndptr corner;
float hi,lo,dx, dy, dz,xx,zz;


qdata = data;
delta = (ymax-ymin)/resln;
for (i=0;i<2292;i++)
  ind[i]=0;
for (i=0;i<num;i++)
  {
   back = 0;
   back2 = 0;
   hi = lo = qdata[4];
   for (j = 1;j<4;j++)
     {
      if (qdata[j*6+4] > hi)
        hi = qdata[j*6+4];
      if (qdata[j*6+4]< lo)
        lo = qdata[j*6+4];
     }
   for (j=0;j<4;j++)
     {
      if (qdata[j*6]>0)
        back++; /* back patch*/
      if (qdata[j*6]<0)
        back2++;  /*front patch*/
      corner.x1=qdata[j*6+3];
      corner.y1=qdata[j*6+4];
      corner.z1=qdata[j*6+5];
      if (j != 3)
       {
```

```
      corner.x2 =qdata[(j+1)*6+3];
      corner.y2 = qdata[(j+1)*6+4];
      corner.z2 = qdata[(j+1)*6+4];
     }
    else
     {
      corner.x2 = qdata[3];
      corner.y2 = qdata[4];
      corner.z2 = qdata[5];
     }
    binum = GetBin(corner.y1,ymin,ymax,delta);
    if (corner.y1 == lo)
      if ((mask[binum].y < corner.y1)&&(binum<resln-1))
        binum++;
    if (corner.y1 == hi)
      if ((mask[binum].y > corner.y1)&&(binum>0))
       binum=binum-1;

    dx = corner.x2 - corner.x1;
    dy = corner.y2 - corner.y1;
    dz = corner.z2 - corner.z1;
    zz = Interp(mask[binum].y-corner.y1,dy,dz)+corner.z1;
    xx = Interp(mask[binum].y-corner.y1,dy,dx)+corner.x1;
    front = mask[binum].b;
    tind=0;
    if (front != NULL)
     while ((xx > front->x)&&(tind==0)&&(front->next != NULL))
       {
       if ((front->zmin <= zz)&&(zz<=front->zmax))
         {
          tind = 1;
          ind[i*4+j]=1;
         }
       else
          front = front->next;
       }
      } /*end of inner for loop*/
     qdata = &qdata[24];
   }

} /*chk_shade*/
```

```
/* ============================================================ */



/* ============================================================ */
void Shadow(pdata,mask,patchnum,ymax,ymin,ind)
/*This procedure reads in the template and output a boolean integer
  for each of the corners in the template.
   input:  pdata, patchnum, ymax, ymin.
   pdata:  template of the target.
   patchnum:  number of patches in the template.
   ymin, ymax:  max and min values of y of the template.
   output:mask, ind
   mask:the shading mask of the template.
   ind: boolean integer indicates whether the corner is shaded.  */
/* ************************************************************ */

float *pdata,ymax,ymin;
int patchnum;
int ind[2292];
msk  *mask;
{
float ygmin,ygmax;
float delta;
float *qdata,*data;
bndptr corner[4];
int  i, patchind,j,k,acc[resln];

    /*printf("Entering Shadow... patch num = %d\n",patchnum);
    printf("ymin = %f, ymax = %f\n",ymin,ymax);*/
    ygmin = ymin;
    ygmax = ymax;
    delta = (ygmax - ygmin)/resln;
    InitMask(mask,ygmin,delta);
    qdata = (float *)malloc(patchnum*24*sizeof(float));
    if (qdata == NULL)
     printf("not enough space for malloc qdata\n");
    for (i = 0;i<(patchnum*24);i++)
      qdata[i]=pdata[i];
    data = qdata;
    for (i=0;i<patchnum;i++)
    {
        GetBounds(qdata,corner);   /* boundaries of patch */
        /* now go through lines setting z-bounds */
        Trace(corner,mask,delta,ygmin,ygmax);
        qdata = &qdata[24];
     }
    qdata = data;
    chk_shade(data,patchnum,ygmin,ygmax,mask,ind);
    data = NULL;
    free(qdata);
}  /* Shadow */
/* =============================================================== */



/* =============================================================== */
void change_extra(data,ymin,ymax,P,r1)
/* This procedure shifts and rotates the template to the proper
   positionand orientation.  Also it finds the minimum and maximum
   y values of the template after shifting and rotation.
   input:  data, P, r1
   data:  the template
   P    :  the translation vector
   r1   :  the rotation matrix.
   output : data, ymin, ymax
   data : the new template.  It replaces the inputed template.
   ymin, ymax: minimum and maximum y values of the output template.*/
/* *************************************************************** */

float *data;
float *ymin,*ymax;
float P[3],r1[3][3];

{
  int i,j,k;
  float v[3];
  float *q;

  q = data;
  for (i=0;i<2292;i++)
   {
    q[3]=q[3]-P[0];
    q[4]=q[4]-P[1];
    q[5]=q[5]-P[2];

    for (j=0;j<2;j++)
      {
       for (k=0;k<3;k++)
         v[k]= q[j*3+k];
       Rota(r1,v);
       for(k=0;k<3;k++)
         q[j*3+k]=v[k];
       }
    if (i == 0)
       *ymin = *ymax = v[1];
    else
      {
       if (v[1] > *ymax)
         *ymax = v[1];
       if (v[1] < *ymin)
         *ymin = v[1];
      }
    q = &q[6];
   }
} /*end of change_extra*/
/* ============================================================ */



/* ============================================================ */
void out(refl,fname)
/* This procedure writes the complex valued reflectivity into a file.
   input: refl, fname;
    refl:  the reflectivity of the template.
    fname: the name of the file to which will be written.
   output:  a file pointed by fname.     */
/* ************************************************************ */

cmplx refl[range_num];
char fname[20];

{
  int i;
  float mag;
  FILE *fp;

  fp = fopen(fname,"w");
  for (i=0;i<range_num;i++)
    fprintf(fp,"%f\t%f\n",refl[i].real,refl[i].imag);
 fclose(fp);
} /*end of out*/
/* =============================================================== */



/* =============================================================== */
void job(pp,n,refl,xgmin,xgmax)
/*Job reads in the template, generates the shading masks, calculates
 the reflectivity, and writes the results to files.  The names of
 the files are self-generated.
/* *************************************************************** */

float *pp;
long n;
cmplx refl[range_num];
float xgmin,xgmax;
{
float *q;
int i,j,skip,ga;
float ygmin,ygmax;
CN corner[4];
float P[3],R[3];
float A[4];
float r1[3][3];
int ind[2292],k;
msk *pmask;
char fname[20];
int cx,cy,cz;
cmplx *phase;
float largest,smallest;

printf("Starting process, please wait....\n");
sprintf(fname, "rp000\0");
```

```
q = (float *)malloc(n*6*sizeof(float));
init(P,3);  /*no shifting */
phase=(cmplx *)malloc(n/4*sizeof(cmplx));
printf("ready to go in the loop\n");
/*for (cx = -9; cx<= 9; cx++)
  for (cy = -18; cy < 18; cy++)*/
   for (cz = 90; cz <= 108; cz++)
       {
         /* beginning of one set */
         randpha(phase,n);
         for (i=0;i<n*6;i++)
            q[i]=pp[i];
         R[0]=0*10*pi/180;
         R[1]=0*10*pi/180;
         R[2]=cz*pi/180;
         cmpinit(refl,range_num);
         pmask = (msk *) malloc (resln*sizeof(msk));
         rmatrix(R,r1);
         change_extra(q,&ygmin,&ygmax,P,r1);
         Shadow(q,pmask,n/4,ygmax,ygmin,ind);
         /* for one patch, iterate for n/4 patches */
         for (i=0;i<(n/4);i++)
          {
           for (j=0;j<4;j++)
            {
             corner[j].xn = q[i*24+j*6];
             corner[j].yn = q[i*24+j*6+1];
             corner[j].zn = q[i*24+j*6+2];
             corner[j].x = q[i*24+j*6+3];
             corner[j].y = q[i*24+j*6+4];
             corner[j].z = q[i*24+j*6+5];
            }

          area(corner,A,ind,i);
          J_area(corner,A,3E8/3.35E9);
          sum_reflect(corner,A,phase[0],xgmin,xgmax,refl);
          /*sum_reflect(corner,A,phase[0],-5.,5.,refl);*/
        }
       covsinc(refl);   /* convolve with a sinc function */

       /******************generating filename ************/
       inc(fname,4);
       out(refl,fname);
       for (ga = 0;ga < resln;ga++)
         free(pmask[ga].b);
       free(pmask);
       /***** up to here, one set of range profile is computed*****/
     }
 free(q);
 free(phase);
} /* end of job */
/* ============================================================== */



/* ============================================================== */
void change_data(pp,n,xgmin,xgmax)
/*change-data scales up or down the fuselage of the original
  template to  10 meters long.
   input:  pp, n.
     pp : template of the target
     n  : the number of patches in the template
   output:  xgmin, xgmax, pp
    pp  : template of the target with the fuselage eqaul to 10
          meters.  It replaces the original template.
  xgmin,xgmax:min and max x values of the template after
               the scaling.                                     */
/* ************************************************************** */

float *pp;
int n;
float *xgmin,*xgmax;
{
 int i;
 float min,max,scale;
 float a,b,c;
 float newdata[3];

 min = max = pp[4];
 for (i=1;i<n;i++)
   {
   if(pp[i*6+4] > max)
```

```
      max = pp[i*6+4];
   if (pp[i*6+4] < min)
      min = pp[i*6+4];
   }
  scale = 10/(max - min);
  for (i=0;i<n;i++)
    {
      newdata[0] = pp[i*6+4]*scale;
      newdata[1] = pp[i*6+3]*scale;
      newdata[2] = -pp[i*6+5]*scale;
      pp[i*6+3] = newdata[0];
      pp[i*6+4] = newdata[1];
      pp[i*6+5] = newdata[2];
    }
  *xgmin = scale * min;
  *xgmax = scale * max;
} /*end of job*/
/* ============================================================== */



/* ============================================================== */
main(argv)
char argv[20];
{
float  *pp;
long n;
cmplx refl[range_num];
float xgmin,xgmax;

argv = "plane.bin\0";
readdata(&pp, &n, argv); /* n is # of corners*/
change_data(pp,n,&xgmin,&xgmax);
job(pp,n,refl,xgmin,xgmax);

}/* end of main*/
/* ============================================================== */
```

8. ACKNOWLEDGEMENTS

This research was supported by the Rome Laboratory in Rome, NY under contract number F30602-92-C-0004. The support for my work is sincerely appreciated.

I would like to thank my thesis advisor, Dr. Joseph O'Sullivan, for his constant advice, consultation, accessibility as well as his encouragement.

I feel very fortunate to be a member of ESSRL. It is amazing that so many highly talented people can work so well together. I also feel very lucky to be one of the "three musketeers". You have been wonderful friends and great support. I will always remember those late nights we worked together. I also want to thank Don, Steve, Nick, Brian, and Eric. You people have been such great friends and office mates. Jolley 424 is the best office environment I ever had. I am also very grateful to all my friends in St. Louis, especially Yan.

Finally, I would like to thank my parents and grandparents. If you had not pushed me that hard when I was little, I probably would not be here today. It is also your loving and caring that helped me overcome any difficulties I experienced. This work is dedicated to you.

9. Bibliography

[1] Bir Bhanu, "Automatic Target Recognition: State of the Art Survey," *IEEE Transactions on Aerospace and Electronic Systems*, vol.AES-22, No. 4, pp. 364, July 1986.

[2] M. Cohen, "Variability of ultra-high range resolution radar profiles and some implications for target recognition," *SPIE*, vol. 1699, pp. 256 - 266, 1992.

[3] C. Smith and P. Goggans, "Radar Target Identification," *IEEE Antennas and Propagation Magazine*, vol. 35, No. 2, pp. 27 -37, April 1993.

[4] D. R. Wehner, *High Resolution Radar*, Artech House, Norwood, Mass., 1987.

[5] H. L. Van Trees, *Detection, Estimation, and Modulation Theory*, Vol. I, pp. 253, Wiley and Sons, New York, 1968.

[6] A. Srivastava, N. Cutaia, M. I. Miller, J. A. O'Sullivan, and D. L. Snyder, "Multi-Target Narrow-band Direction Finding and Tracking Based on Motion Dynamics," *submitted for publication in Electronic Systems and Signals Research Monograph*, Department of Electrical Engineering, Washington University, St. Louis, MO 63130, July 1992.

[7] A. Srivastava, M. I. Miller, and U. Grenander, "Jump-Diffusion Processes for Object Tracking and Direction Finding," *Proceedings 29th Annual Allerton Conference on Communication, Control, and Computing*, pp. 563-570, University of Illinois, Urbana-Champaign, 1991.

[8] H. Anton and C. Rorres, *Elementary Lineary Algebra with Applications*, John Wiley & Sons, Inc., New York, 1987.

[9] A. K. Bhattacharyya, and D. L. Sengupta, *Radar Cross Section Analysis & Control*, Artech House, Norwood, Mass., 1991.

[10] D. J. Andersh, S. W. Lee, R. Pinto, C. Yu, and F. Beckner, "Range Profile Synthesis for Noncooperative Target Identification," preprint.

[11] W. Siebert, *Circuits, Signals, and Systems*, pp. 563, McGraw-Hill Book Company, New York, 1986.

[12] J. A. O'Sullivan, K. C. Du, R. S. Teichman, M. I. Miller, D. L. Snyder, and V. C. Vannicola, "Radar Target Recognition Using Shape Models," *Proceedings 30th Allerton Conference on Communication, Control, and Computing*, pp. 515-523, University of Illinois, Urbana, IL, October 1992.

[13] J. A. O'Sullivan, K. C. Du, R. S. Teichman, M. I. Miller, D. L. Snyder, and V. C. Vannicola, "Reflectivity Models for Radar Target Recognition," *SPIE Proceedings*, vol. 1960, Orlando, FL, April 1993.

[14] B. Friedland, *Control System Design: An Introduction to State-Space Methods*, McGraw-Hill, New York, 1986.

[15] D. L. Snyder, J. A. O'Sullivan, and M. I. Miller, "The Use of Maximum-Likelihood Estimation for Forming Images of Diffuse Radar-Targets from Delay-Doppler Data," *IEEE Transactions on Information Theory*, vol. IT-35, pp. 536-548, 1989.

[16] W. H. Press, et al. *Numerical Recipes in C: The Art of Scientific Computation*, Cambridge University Press, Cambridge, 1988.

[17] D. L. Snyder, J. A. O'Sullivan, and M. I. Miller, "The use of maximum-likelihood estimation for forming images of diffuse radar-targets from delay-doppler data," *IEEE Transactions on Information Theory*, May 1989.

10. Vita

K. Cecilia Du

PII Redacted

[redacted]	[redacted]
[redacted]	[redacted]
Undergraduate Study:	Washington University, St. Louis, Missouri B. S., May, 1992
Graduate Study:	Washington University, St. Louis, Missouri M. S., August, 1993
Professional Societies:	Eta Kappa Nu
Honors/Awards:	Washington University International Student Scholarship (1988 - 1992)
Publications:	See Bibliography items [12, 13]

December, 1993

Short title: **RANGE PROFILE PREDICTION** Du, M.S.E.E. 1993

☆U.S. GOVERNMENT PRINTING OFFICE: 1997-509-127-47100

MISSION
OF
ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.